1 PRODUCT OVERVIEW

INTRODUCTION

Samsung KS17C4000 16/32-bit RISC microcontroller is a cost-effective and high-performance microcontroller solution for DVD and general purpose applications.

Among the outstanding features of the KS17C4000 is its CPU core, a 16/32-bit RISC processor (ARM7TDMI) designed by Advanced RISC Machines, Ltd. The ARM7TDMI core is a low-power, general-purpose microprocessor macro-cell that was developed for use in application-specific and customer-specific integrated circuits. Its simple, elegant, and fully static design is particularly suitable for cost-sensitive and power-sensitive applications.

The KS17C4000 was developed using the ARM7TDMI core, CMOS standard cell, and a data path compiler. Most of the on-chip function blocks were designed using an HDL synthesizer. The KS17C4000 has been fully verified in Samsung ASIC test environment.

By providing a complete set of common system peripherals, the KS17C4000 minimizes overall system costs and eliminates the need to configure additional components.

The integrated on-chip functions that are described in this document include:

- Memory manager: ROM/SRAM controller, DRAM controller (4 Memory banks)
- Instruction/Data cache and controller (This area is contained to the internal SRAM area)
- Two-channel DMA controller
- 4-Kbyte internal SRAM for stack, data memory, code memory, or cache memory
- One UART and Two SIOs (Synchronous serial I/O)
- Three 16-bit timers and Two 8-bit timers
- Crystal/Ceramic oscillator or external clock can be used as the clock source
- Standby mode support: STOP and IDLE mode
- One 8-bit basic timer and 3-bit watchdog timer
- Interrupt controller (32 interrupt sources)
- Eight 8-bit ADCs
- Eight programmable I/O ports and One extra output extension port control pin
- 100-pin QFP/TQFP, 128-pin QFP

FEATURES

Architecture

- Completely integrated system for embedded applications, especially DVD applications
- Complete 16/32-bit RISC architecture
- Efficient and powerful ARM7TDMI CPU core
- Cost effective JTAG-based debugging solution
- Instruction/Data cache (Two Cache size is configured in the internal SRAM area by software)

Memory

- 8-bit, or 16-bit external bus support for ROM/SRAM, and DRAM bank, and external one output control
- Programmable memory access times (from 0 to 7 wait cycles)
- Programmable DRAM refresh controller
- 4-Kbyte SRAM (for stack, data memory, code memory, or cache memory)
- Cost-effective memory-to-peripheral interface signals

DMA

- Two-channel general-purpose DMA controller
- Memory-to-memory, serial port-to-memory, memory-to-serial port data transfers without CPU intervention
- Initiated by software or external DMA request
- Increments or decrements source or destination addresses and supports 8-bit (byte), 16-bit (half word), or 32-bit (word) data transfers

ADC

- Can be used as General purpose Input Ports
- Eight 8-bit resolution channels
- Successive approximation conversion

UART and SIO

- One UART and two synchronous SIOs with DMA-based or interrupt-based operation
- Programmable baud rates
- Supports serial data transmit/receive operations with 5-bit, 6-bit, 7-bit, 8-bit in UART or 8-bit in sync SIO

8-bit timer/counters (T3, T4)

- Two programmable 8-bit timer/counters
- Interval, capture, or PWM mode operation
- Internal or external clock source

16-bit timer/counters (T0, T1, T2)

- Three programmable 16-bit timer/counters
- Interval or capture, match & overflow mode operation
- Internal or external clock source

Basic timer and Watchdog timer

- 8-bit counter + 3-bit counter
- Overflow signal of 8-bit counter makes a basic timer interrupt and controls the oscillation warm-up time
- Overflow signal of 3-bit counter makes a system reset.

External Interface mode

- 24 expandable address pin: 16MB space (A0-A15, D0-D7, A16/D8/AD8-A23/D15/AD15)
- ROM/SRAM/DRAM (4 Banks): Totally 32MB space

I/O Ports

- 8 programmable I/O ports (62 I/O pins)
- Each port pin can be configured individually as input, output, or I/O for a dedicated signal
- 8-bit or 4-bit real time output (P0/RP)

Interrupts

- 32 interrupt sources (External Interrupt: 13)
- Normal or fast interrupt modes (IRQ, FIQ)

Power down mode

- IDLE and STOP
- System clock control (1/1,1/2, 1/8, 1/16, and 1/1024)

Operating Voltage Range

- 2.7 V to 5.5 V

Package Type

- 100-pin QFP/TQFP, or 128-pin QFP

BLOCK DIAGRAM

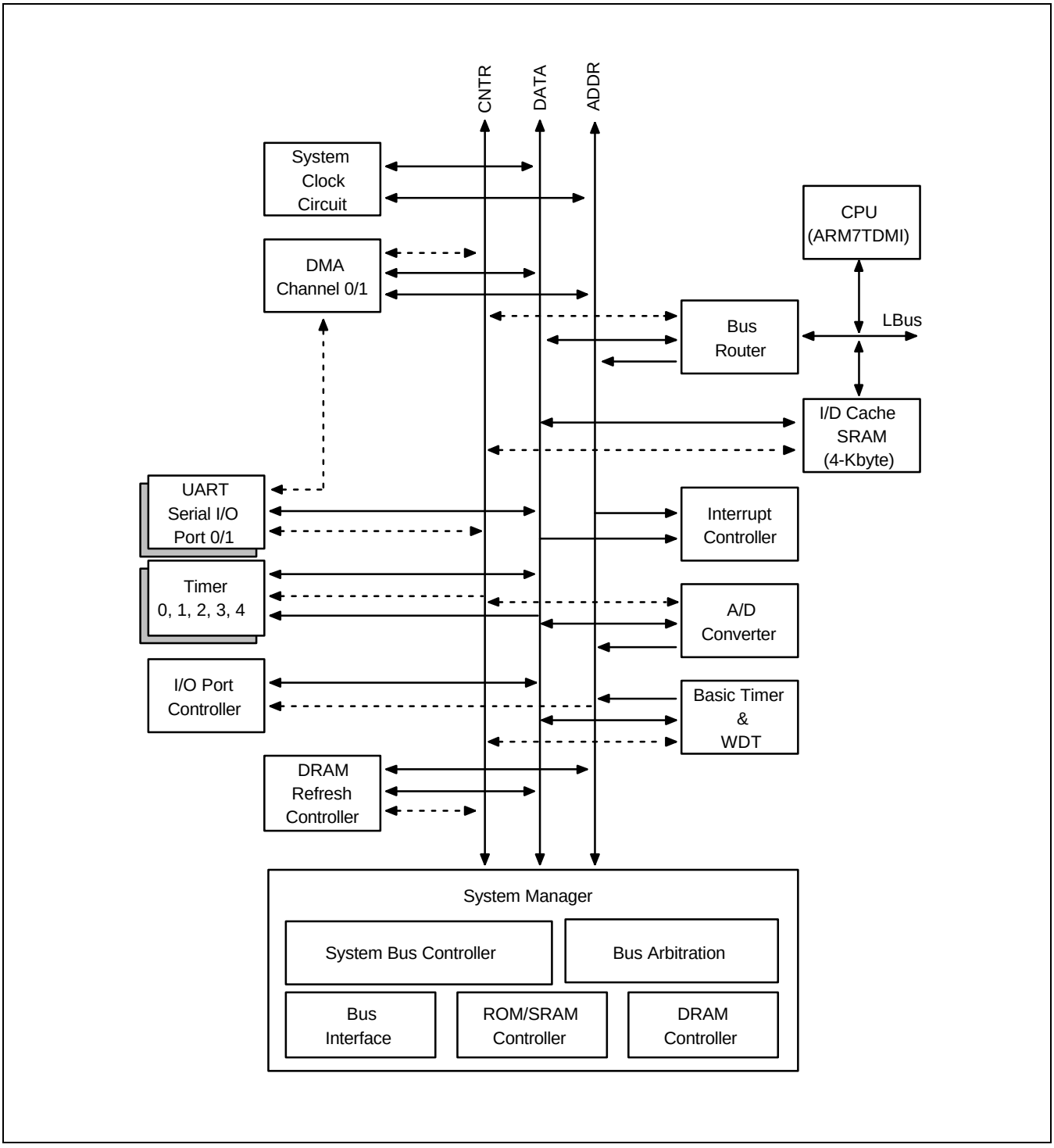


Figure 1-1. KS17C4000 Block Diagram

PIN ASSIGNMENTS

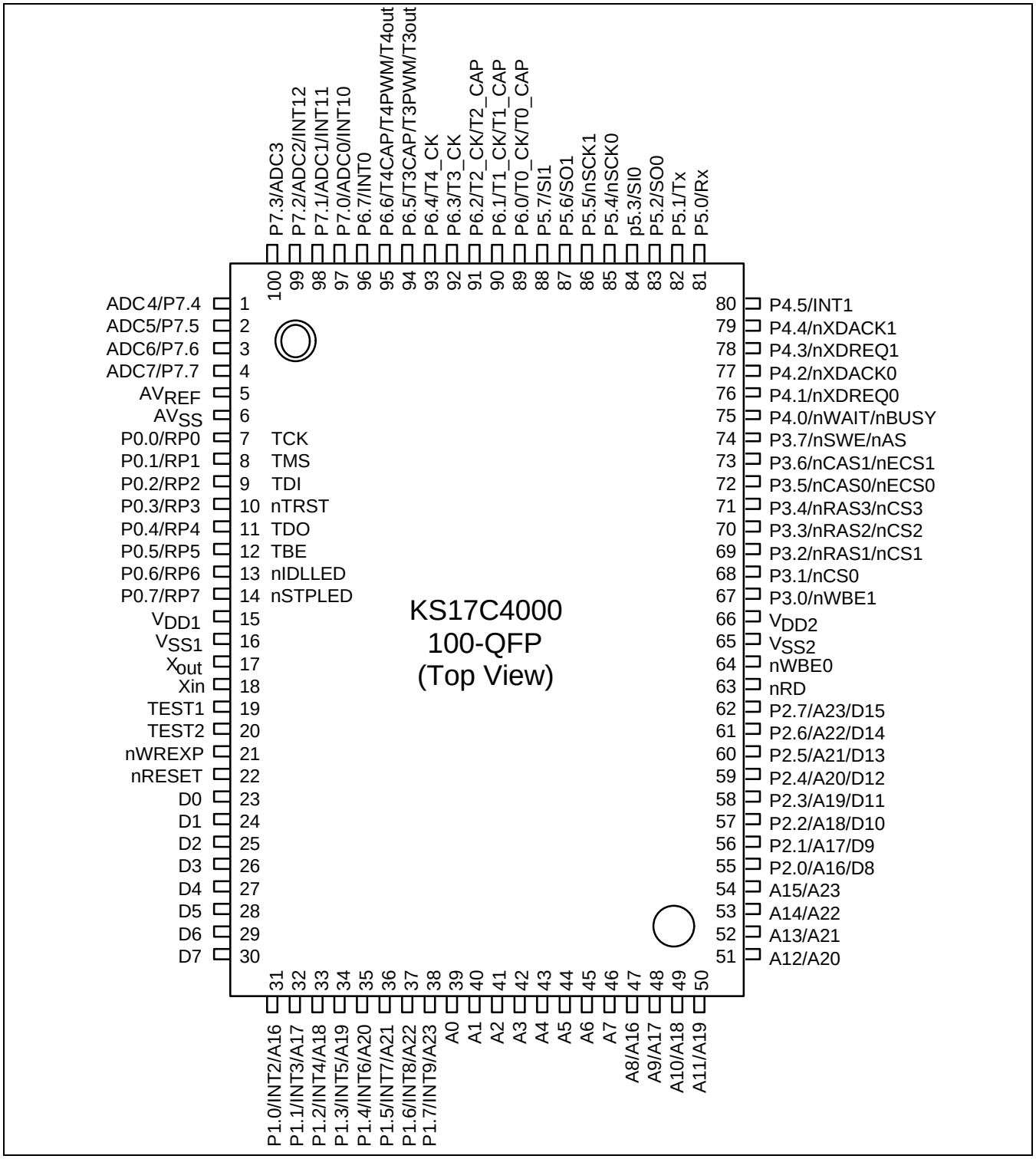


Figure 1-2. KS17C4000 Pin Assignments (100-QFP)

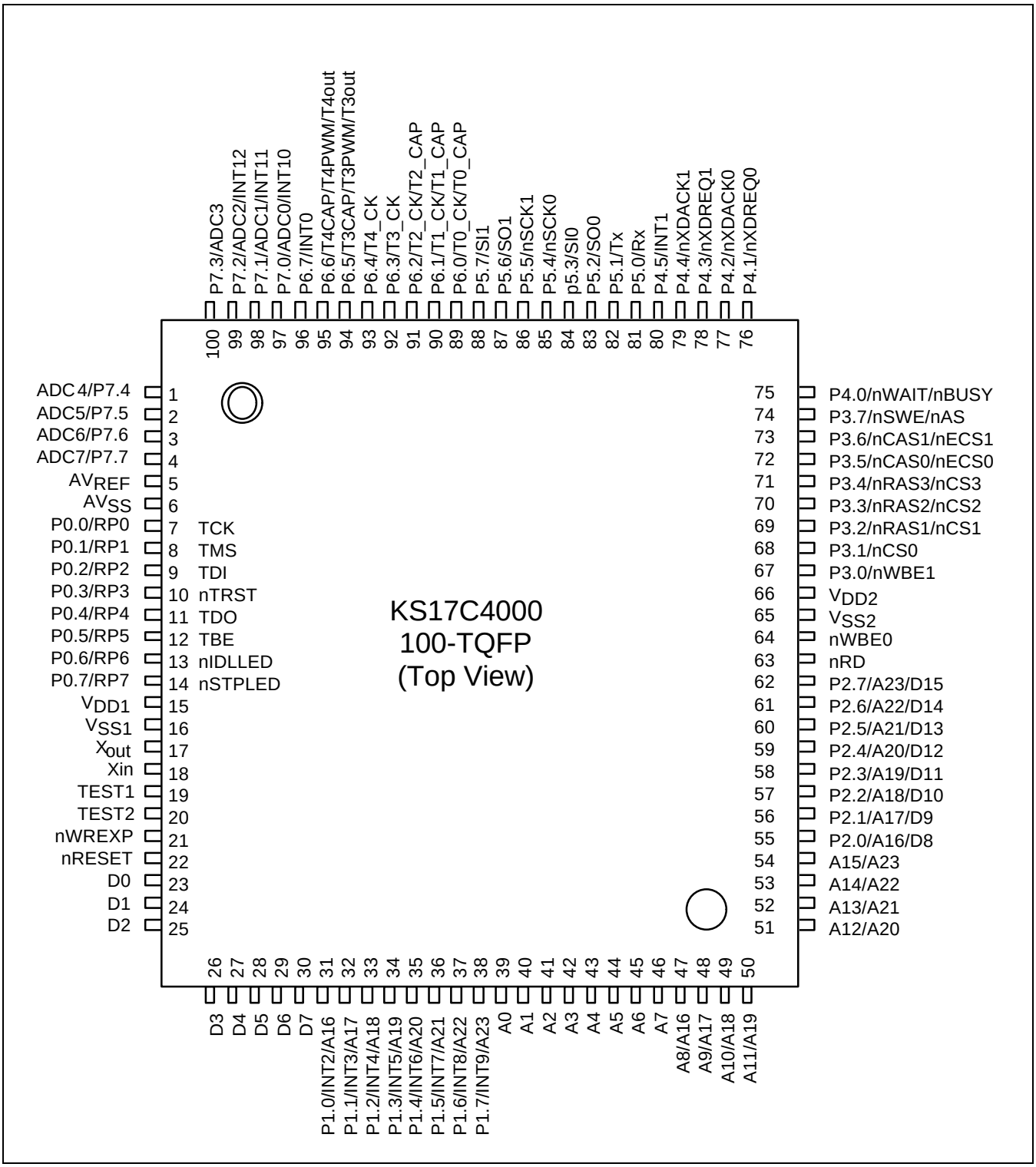


Figure 1-3. KS17C4000 Pin Assignments (100-TQFP)

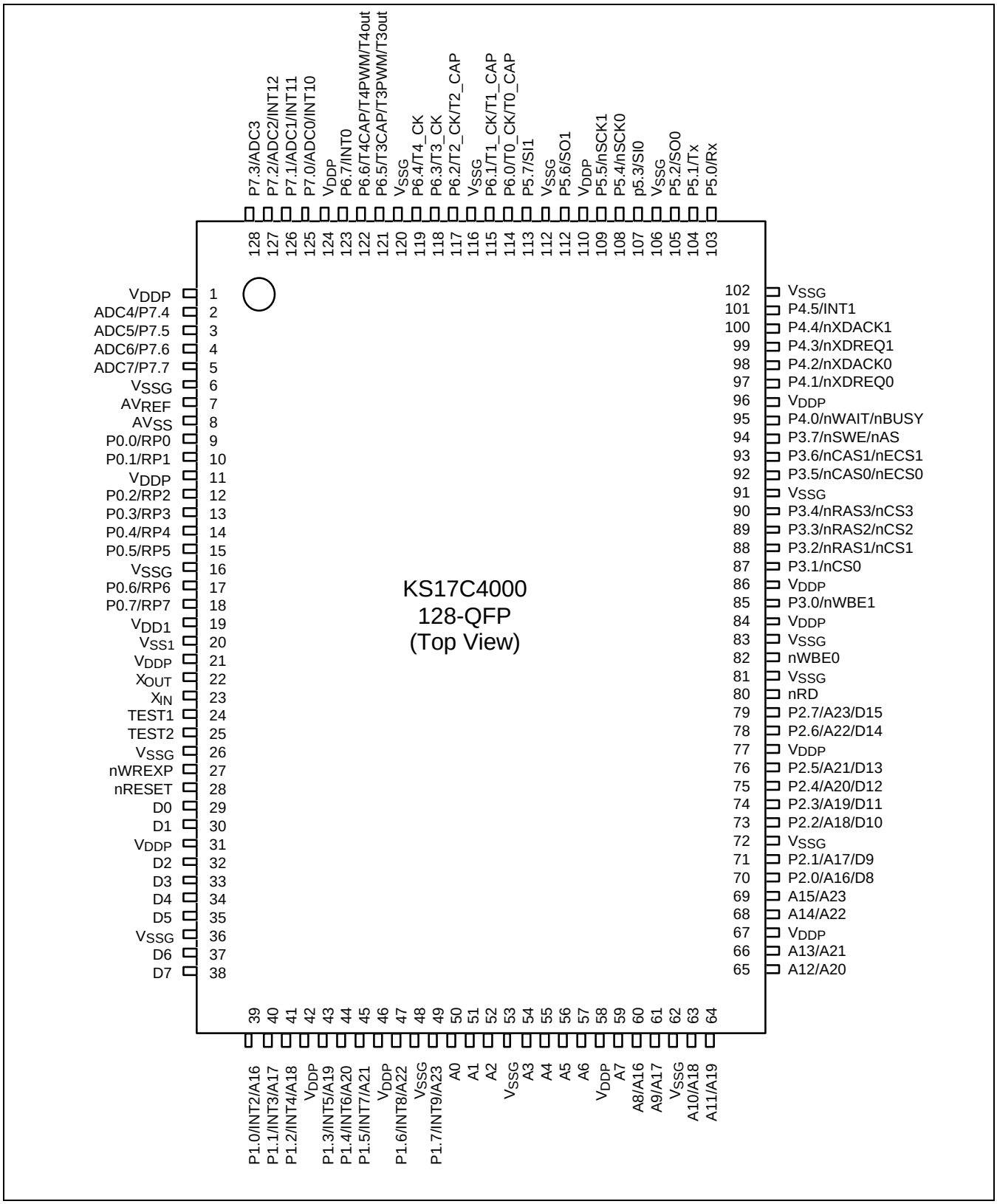


Figure 1-4. KS17C4000 Pin Assignments (128-QFP)

SIGNAL DESCRIPTIONS

Table 1-1. KS17C4000 Signal Descriptions (100-pin QFP)

Signal Name	Pin Number	I/O Pin Type	Description
P0.0/RP0-P0.7/RP7	7-14	0	Bit programmable I/O ports for CMOS input or output. The eight (P0 only) or sixteen (P0 and P1) I/O ports can be configured to control a read or write command. Port 0 can be used as a real time output (RP0-RP7); in an 8-bit or 4-bit unit. In MDS mode, Port 0 is changed to Test Access Port for JTAG and emulated by external port (EXTPORT) and external output latch. But the emulated port 0 should be used only as an output port, not an input port.
Xout, Xin	17, 18	–	The KS17C4000 master clock. It has a 50% duty cycle and operating frequency of 20-33 MHz.
nRESET	22	9	nRESET is the global reset input for the KS17C4000. For a system reset, nRESET must be held to Low level for at least 128 machine cycles.
D0-D7	23-30	8	Data bus: D0-D7
P1.0/INT2/A16-P1.7/INT9/A23	31-38	1	Bit programmable I/O port for Schmitt trigger input or push-pull output. P1 can alternatively be used as external interrupt lines, INT2-INT9 or address bus lines, A16-A23 for external interface. P1.0/INT2/A16: External interrupt input 2, or address line A16 P1.1/INT3/A17: External interrupt input 3, or address line A17 P1.2/INT4/A18: External interrupt input 4, or address line A18 P1.3/INT5/A19: External interrupt input 5, or address line A19 P1.4/INT6/A20: External interrupt input 6, or address line A20 P1.5/INT7/A21: External interrupt input 7, or address line A21 P1.6/INT8/A22: External interrupt input 8, or address line A22 P1.7/INT9/A23: External interrupt input 9, or address line A23
A0-A7	39-46	8-1	Address bus: A0-A7
A8/A16-A15/A23	47-54	8-2	Address bus: A8-A15 or A8/A16-A15/A23
P2.0/A16/D8-P2.7/A23/D15	55-62	2	Bit programmable I/O port for CMOS input or push-pull output port. P2 can alternatively be used as external or address lines (higher), A16-A23, or data (higher) lines, D8-D15. P2.0-P2.7: Bit assignable normal I/O port A16-A23: Bit assignable Address line output D8-D15: Data line I/O port
nRD	63	8-1	Reading strobe signal for external memory
nWBE0	64	8-1	Writing strobe signal for data on pins D0 to D7

NOTE: P1, P2, and P3.1 are in the low state by pull-down resistor and input mode after a reset release. Therefore, external circuitry should be designed in such a way that output of a low-level signal in the initial state causes no problem.

Table 1-1. KS17C4000 Signal Descriptions (Continued)

Signal Name	Pin Number	I/O Pin Type	Description
P3.0/nWBE1-P3.7/nAS	67-74	3 3-1	Bit programmable I/O port for CMOS input or push-pull output. P3 can alternatively be used as an external interface control line. P3.0/nWBE1: High writing strobe signal for data on pins D8 to D15 P3.1/nCS0: nCS0: chip select strobe when the address is within the specified address area. P3.2/nRAS1/nCS1: nRAS1: row address strobes for dynamic RAM (DRAM) when the address is within the specified address area. nCS1: chip select strobe when the address is within the specified address area. P3.3/nRAS2/nCS2: nRAS2: row address strobes for dynamic RAM (DRAM) when the address is within the specified address area. nCS2: chip select strobe when the address is within the specified address area. P3.4/nRAS3/nCS3: nRAS3: row address strobes for dynamic RAM (DRAM)when the address is within the specified address area. nCS3: chip select strobe when the address is within the specified address area. P3.5/nCAS0/nECS0: nCAS0: column address strobes for the dynamic RAM (DRAM) bank. nECS0: chip select address strobes for the extra device P3.6/nCAS1/nECS1: nCAS1: column address strobes for the dynamic RAM (DRAM) bank. nECS1: chip select address strobes for the extra device P3.7/nSWE/nAS: x16 SRAM write enable or Address strobe.
P4.0/nWAIT/nBUSY-P4.5/INT1	75-80	4 4-1	Bit programmable I/O port for C-MOS input or push-pull output. P4.1, 4.2, 4.3, and 4.4 can alternatively be used as DMA control line, and P4.0 can be used as external interface wait input pin. P4.0 / nWAIT / nBUSY: Wait signal for requesting CPU bus wait. nBusy - Busy signal for SIO operation. P4.1 / nXDREQ0 P4.3 / nXDREQ1: External DMA request. nXDREQ is asserted by a peripheral device to request a data transfer using DMAC. This signal must be held active for at least four machine cycles. P4.2 / nXDACK0 P4.4 / nXDACK1: DMA acknowledge. This active Low output signal is generated whenever a DMA transfer is completed. P4.5 / INT1: External interrupt input 1.

NOTE: P3.0, P3.2, 3.3, 3.4, 3.5, 3.6, and P3.7 are in the high state by pull-up resistor and input mode after a reset release. Therefore, external circuitry should be designed in such a way that output of a high-level signal in the initial state causes no problem.

Table 1-1. KS17C4000 Signal Descriptions (Continued)

Signal Name	Pin Number	I/O Pin Type	Description
P5.0/Rx- P5.7/SI1	81-88	5	Bit programmable I/O port for Schmitt trigger input or push-pull output. P5 can alternatively be used as multiplexed pins. P5.0/Rx: Rx Data input for the UART P5.1/Tx: Tx Data output for the UART P5.2/SO0: Serial data output for the SIO 0. P5.3/SI0: Serial data input for the SIO 0. P5.4/nSCK0: Serial clock input/output for the SIO 0 or clock input for an UART. P5.5/nSCK1: Serial clock input/output for the SIO 1. P5.6/SO1: Serial data output for the SIO 1. P5.7/SI1: Serial data input for the SIO 1.
P6.0/T0- P6.7/INT0	89-96	6 4-1	Bit programmable I/O port for Schmitt trigger input or push-pull output. P6 can alternatively be used as a multiplexed pin. P6.0/T0: Timer 0 clock or capture input P6.1/T1: Timer 1 clock or capture input P6.2/T2: Timer 2 clock or capture input P6.3/T3: Timer 3 clock input P6.4/T4: Timer 4 clock input P6.5/T3: Timer 3 capture input or PWM/Toggle Out output P6.6/T4: Timer 4 capture input or PWM/Toggle Out output P6.7/INT0: External interrupt input 0.
P7.0/ADC0/INT 10-P7.7/ADC7	97-4	7 7-1	Bit programmable input port for input or analog input. P7 can alternatively be used as analog-to-digital input pins P7.0/ADC0/INT10: Analog input 0 or external interrupt INT10 P7.1/ADC1/INT11: Analog input 1 or external interrupt INT11 P7.2/ADC2/INT12: Analog input 2 or external interrupt INT12 P7.3/ADC3: Analog input 3 P7.4/ADC4: Analog input 4 P7.5/ADC5: Analog input 5 P7.6/ADC6: Analog input 6 P7.7/ADC7: Analog input 7
Avref, Avss	5, 6	–	Power supply pin and GND pin for A/D converter
nWREXP	21	8-1	External port (EXTPORT) write strobe signal. In normal operating mode, this pin indicates as write strobe signal for external port consisted of 8-bit or 16-bit latch. In MDS mode, this pin indicates as a write strobe signal for external 8-bit latch device for port 0 output emulation.
TEST1, 2	19, 20	–	Test pin for manufacturing. Test 2 Test 1 0 0 : Normal mode, 8bit data bus mode at bank 0. 0 1 : Normal mode, 16bit data bus mode at bank 0. 1 0 : MDS mode, 8bit data bus mode at bank 0 1 1 : MDS mode, 16bit data bus mode at bank 0. (P0.0 → TCK, P0.1 → TMS, P0.2 → TDI, P0.3 → nTRST, P0.4 → TDO, P0.5 → TBE, P0.6 → nIDLLED, P0.7 → nSTPLED)
VDD1, VSS1, VDD2, VSS2	15, 16, 66, 65	–	Power supply pins

CPU CORE OVERVIEW

The KS17C4000 CPU core is the ARM7TDMI processor, a general purpose, 32-bit microprocessor developed by Advanced RISC Machines, Ltd. (ARM). The core's architecture is based on Reduced Instruction Set Computer (RISC) principles. The RISC architecture makes the instruction set and its related decoding mechanisms simpler and more efficient than with microprogrammed Complex Instruction Set Computer (CISC) systems. The resulting benefit is high instruction throughput and impressive real-time interrupt response. Pipelining is also employed so that all components of the processing and memory systems can operate continuously. The ARM7TDMI has a 32-bit address bus.

An important feature of the ARM7TDMI processor, and one which differentiates it from the ARM7 processor, is a unique architectural strategy called *THUMB*. The THUMB strategy is an extension of the basic ARM architecture and consists of 36 instruction formats. These formats are based on the standard 32-bit ARM instruction set, but have been re-coded using 16-bit wide opcodes.

Because THUMB instructions are one-half the bit width of normal ARM instructions, they produce very high-density code. When a THUMB instruction is executed, its 16-bit opcode is decoded by the processor into its equivalent instruction in the standard ARM instruction set. The ARM core then processes the 16-bit instruction as it would a normal 32-bit instruction. In other words, the Thumb architecture gives 16-bit systems a way to access the 32-bit performance of the ARM core without incurring the full overhead of 32-bit processing.

Because the ARM7TDMI core can execute both standard 32-bit ARM instructions and 16-bit Thumb instructions, it lets you mix routines of Thumb instructions and ARM code in the same address space. In this way, you can adjust code size and performance, routine by routine, to find the best programming solution for a specific application.

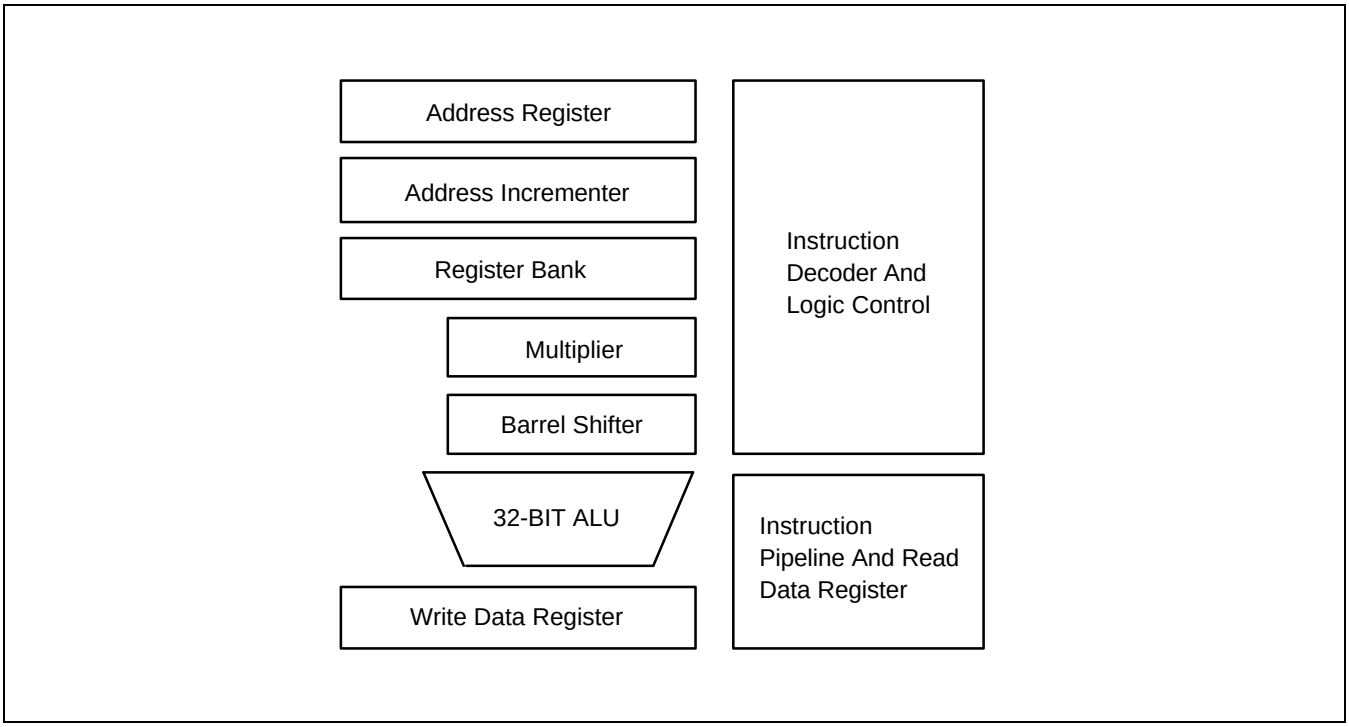


Figure 1-5. ARM7TDMI Core Block Diagram

INSTRUCTION SET

The KS17C4000 instruction set is divided into two subsets: a standard *32-bit ARM instruction set* and a *16-bit THUMB instruction set*.

The 32-bit ARM instruction set is comprised of thirteen basic instruction types which can, in turn, be divided into four broad classes:

- Four types of branch instructions which control program execution flow, instruction privilege levels, and switching between ARM code and THUMB code.
- Three types of data processing instructions which use the on-chip ALU, barrel shifter, and multiplier to perform high-speed data operations in a bank of 31 registers (all with 32-bit register widths).
- Three types of load and store instructions which control data transfer between memory locations and the registers. One type is optimized for flexible addressing, another for rapid context switching, and the third for swapping data.
- Three types of co-processor instructions which are dedicated to controlling external co-processors. These instructions extend the off-chip functionality of the instruction set in an open and uniform way.

NOTE

All 32-bit ARM instructions can be executed conditionally.

The 16-bit THUMB instruction set contains 36 instruction formats drawn from the standard 32-bit ARM instruction set. The THUMB instructions can be divided into four functional groups:

- Four branch instructions.
- Twelve data processing instructions, which are a subset of the standard ARM data processing instructions.
- Eight load and store register instructions.
- Four load and store multiple instructions.

NOTE

Each 16-bit THUMB instruction has a corresponding 32-bit ARM instruction with the identical processing model.

The 32-bit ARM instruction set and the 16-bit THUMB instruction sets are good targets for compilers of many different high-level languages. When assembly code is required for critical code segments, the ARM programming technique is straightforward, unlike that of some RISC processors which depend on sophisticated compiler technology to manage complicated instruction interdependencies.

Pipelining is employed so that all parts of the processor and memory systems can operate continuously. Typically, while one instruction is being executed, its successor is being decoded, and a third instruction is being fetched from memory.

MEMORY INTERFACE

The CPU memory interface has been designed to allow the highest performance potential to be realized without incurring high costs in the memory system. Speed-critical control signals are pipelined so that system control functions can be implemented in standard low-power logic. These pipelined control signals let you fully exploit the fast local access modes offered by industry standard dynamic RAMs.

OPERATING STATES

From a programmer's point of view, the ARM7TDMI core is always in one of two operating states. These states, which can be switched by software or by exception processing, are:

- *ARM state* (when executing 32-bit, word-aligned, ARM instructions), and
- *THUMB state* (when executing 16-bit, half-word aligned THUMB instructions).

OPERATING MODES

The ARM7TDMI core supports seven operating modes:

- *User mode*: the normal program execution state
- *FIQ (Fast Interrupt Request) mode*: for supporting a specific data transfer or channel process
- *IRQ (Interrupt ReQuest) mode*: for general purpose interrupt handling
- *Supervisor mode*: a protected mode for the operating system
- *Abort mode*: entered when a data or instruction pre-fetch is aborted
- *System mode*: a privileged user mode for the operating system
- *Undefined mode*: entered when an undefined instruction is executed

Operating mode changes can be controlled by software, or they can be caused by external interrupts or exception processing. Most application programs execute in User mode. Privileged modes (that is, all modes other than User mode) are entered to service interrupts or exceptions, or to access protected resources.

REGISTERS

The KS17C4000 CPU core has a total of 37 registers: 31 general-purpose, 32-bit registers, and 6 status registers. Not all of these registers are always available. Which registers are available to the programmer at any given time depends on the current processor operating state and mode.

NOTE

When the KS17C4000 is operating in ARM state, 16 general registers and one or two status registers can be accessed at any time. In privileged mode, mode-specific banked registers are switched in.

Two register sets, or banks, can also be accessed, depending on the core's current state: the *ARM state register set* and the *THUMB state register set*:

- The ARM state register set contains 16 directly accessible registers: R0-R15. All of these registers, except for R15, are for general-purpose use, and can hold either data or address values. An additional (seventeenth) register, the CPSR (Current Program Status Register), is used to store status information.
- The THUMB state register set is a subset of the ARM state set. You can access eight general registers, R0-R7, as well as the program counter (PC), a stack pointer register (SP), a link register (LR), and the CPSR. Each privileged mode has a corresponding banked stack pointer, link register, and saved process status register (SPSR).

The THUMB state registers are related to the ARM state registers as follows:

- THUMB state R0-R7 registers and ARM state R0-R7 registers are identical
- THUMB state CPSR and SPSRs and ARM state CPSR and SPSRs are identical
- THUMB state SP, LR, and PC map directly to ARM state registers R13, R14, and R15, respectively

In THUMB state, registers R8-R15 are not part of the standard register set. However, you can access them for assembly language programming and use them for fast temporary storage, if necessary.

EXCEPTIONS

An *exception* arises whenever the normal flow of program execution is interrupted. For example, when processing must be diverted to handle an interrupt from a peripheral. The processor's state just prior to handling the exception must be preserved so that the program flow can be resumed when the exception routine is completed. Multiple exceptions may arise simultaneously.

To process exceptions, the KS17C4000 uses the banked core registers to save the current state. The old PC value and the CPSR contents are copied into the appropriate R14 (LR) and SPSR register. The PC and mode bits in the CPSR are forced to a value which corresponds to the type of exception being processed.

The KS17C4000 core supports seven types of exceptions. Each exception has a fixed priority and a corresponding privileged processor mode, as shown in Table 1-2.

Table 1-2. KS17C4000 CPU Exceptions

Exception	Mode on Entry	Priority
Reset	Supervisor mode	1 (Highest)
Data abort	Abort mode	2
FIQ	FIQ mode	3
IRQ	IRQ mode	4
Prefetch abort	Abort mode	5
Undefined instruction	Undefined mode	6 (Lowest)
SWI	Supervisor mode	6

2 PROGRAMMER'S MODEL

OVERVIEW

KS17C4000 was developed using the advanced ARM7TDMI core designed by Advanced RISC Machines, Ltd.

PROCESSOR OPERATING STATES

From the programmer's point of view, the ARM7TDMI can be in one of two states:

- ARM state which executes 32-bit, word-aligned ARM instructions.
- *THUMB state*which operates with 16-bit, halfword-aligned THUMB instructions. In this state, the PC uses bit 1 to select between alternate halfwords.

NOTE

Transition between these two states does not affect the processor mode or the contents of the registers.

SWITCHING STATE

Entering THUMB State

Entry into THUMB state can be achieved by executing a BX instruction with the state bit (bit 0) set in the operand register.

Transition to THUMB state will also occur automatically on return from an exception (IRQ, FIQ, UNDEF, ABORT, SWI etc.), if the exception was entered with the processor in THUMB state.

Entering ARM State

Entry into ARM state happens:

- On execution of the BX instruction with the state bit clear in the operand register.
- On the processor taking an exception (IRQ, FIQ, RESET, UNDEF, ABORT, SWI etc.). In this case, the PC is placed in the exception mode's link register, and execution commences at the exception's vector address.

MEMORY FORMATS

ARM7TDMI views memory as a linear collection of bytes numbered upwards from zero. Bytes 0 to 3 hold the first stored word, bytes 4 to 7 the second and so on. ARM7TDMI can treat words in memory as being stored either in Big-Endian or Little-Endian format.

NOTE

The KS17C4000 is configured to the big-endian format.

BIG-ENDIAN FORMAT

In Big-Endian format, the most significant byte of a word is stored at the lowest numbered byte and the least significant byte at the highest numbered byte. Byte 0 of the memory system is therefore connected to data lines 31 through 24.

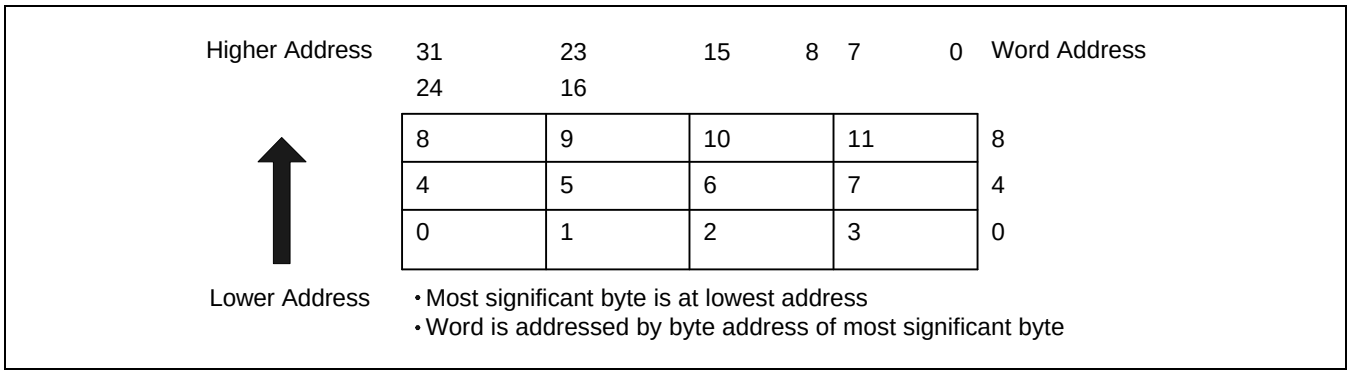


Figure 2-1. Big-Endian Addresses of Bytes within Words

NOTE

The data locations in the external memory are different with Figure 2-1 in the KS17C4000. Please refer to the chapter 4, System Manager.

LITTLE-ENDIAN FORMAT

In Little-Endian format, the lowest numbered byte in a word is considered the word's least significant byte, and the highest numbered byte the most significant. Byte 0 of the memory system is therefore connected to data lines 7 through 0.

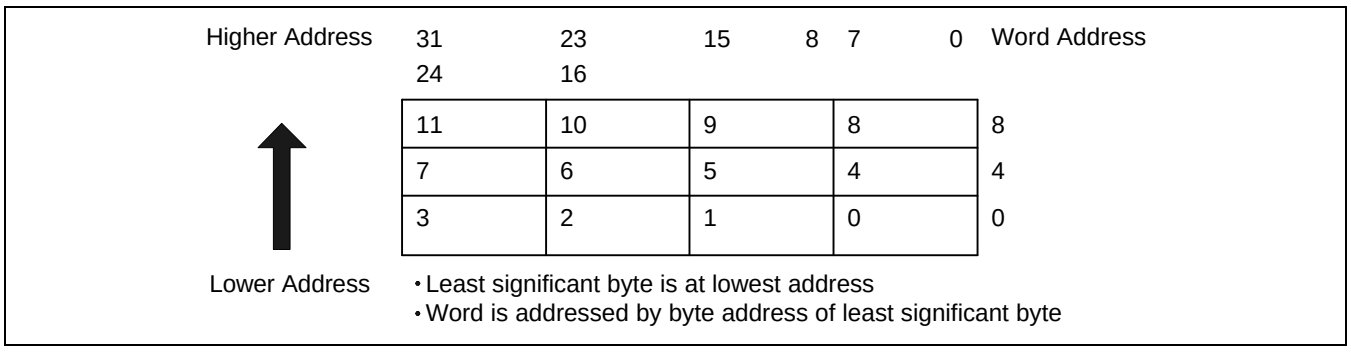


Figure 2-2. Little-Endian Addresses of Bytes whthin Words

INSTRUCTION LENGTH

Instructions are either 32 bits long (in ARM state) or 16 bits long (in THUMB state).

Data Types

ARM7TDMI supports byte (8-bit), halfword (16-bit) and word (32-bit) data types. Words must be aligned to four-byte boundaries and half words to two-byte boundaries.

OPERATING MODES

ARM7TDMI supports seven modes of operation:

- User (usr): The normal ARM program execution state
- FIQ (fiq): Designed to support a data transfer or channel process
- IRQ (irq): Used for general-purpose interrupt handling
- Supervisor (svc): Protected mode for the operating system
- Abort mode (abt): Entered after a data or instruction prefetch abort
- System (sys): A privileged user mode for the operating system
- Undefined (und): Entered when an undefined instruction is executed

Mode changes may be made under software control, or may be brought about by external interrupts or exception processing. Most application programs will execute in User mode. The non-user modes' known as privileged modes-are entered in order to service interrupts or exceptions, or to access protected resources.

REGISTERS

ARM7TDMI has a total of 37 registers - 31 general-purpose 32-bit registers and six status registers - but these cannot all be seen at once. The processor state and operating mode dictate which registers are available to the programmer.

The ARM State Register Set

In ARM state, 16 general registers and one or two status registers are visible at any one time. In privileged (non-User) modes, mode-specific banked registers are switched in. Figure 2-3 shows which registers are available in each mode: the banked registers are marked with a shaded triangle.

The ARM state register set contains 16 directly accessible registers: R0 to R15. All of these except R15 are general-purpose, and may be used to hold either data or address values. In addition to these, there is a seventeenth register used to store status information.

Register 14	is used as the subroutine link register. This receives a copy of R15 when a Branch and Link (BL) instruction is executed. At all other times it may be treated as a general-purpose register. The corresponding banked registers R14_svc, R14_irq, R14_fiq, R14_abt and R14_und are similarly used to hold the return values of R15 when interrupts and exceptions arise, or when Branch and Link instructions are executed within interrupt or exception routines.
Register 15	holds the Program Counter (PC). In ARM state, bits [1:0] of R15 are zero and bits [31:2] contain the PC. In THUMB state, bit [0] is zero and bits [31:1] contain the PC.
Register 16	is the CPSR (Current Program Status Register). This contains condition code flags and the current mode bits.

FIQ mode has seven banked registers mapped to R8-14 (R8_fiq-R14_fiq). In ARM state, many FIQ handlers do not need to save any registers. User, IRQ, Supervisor, Abort and Undefined each have two banked registers mapped to R13 and R14, allowing each of these modes to have a private stack pointer and link registers.

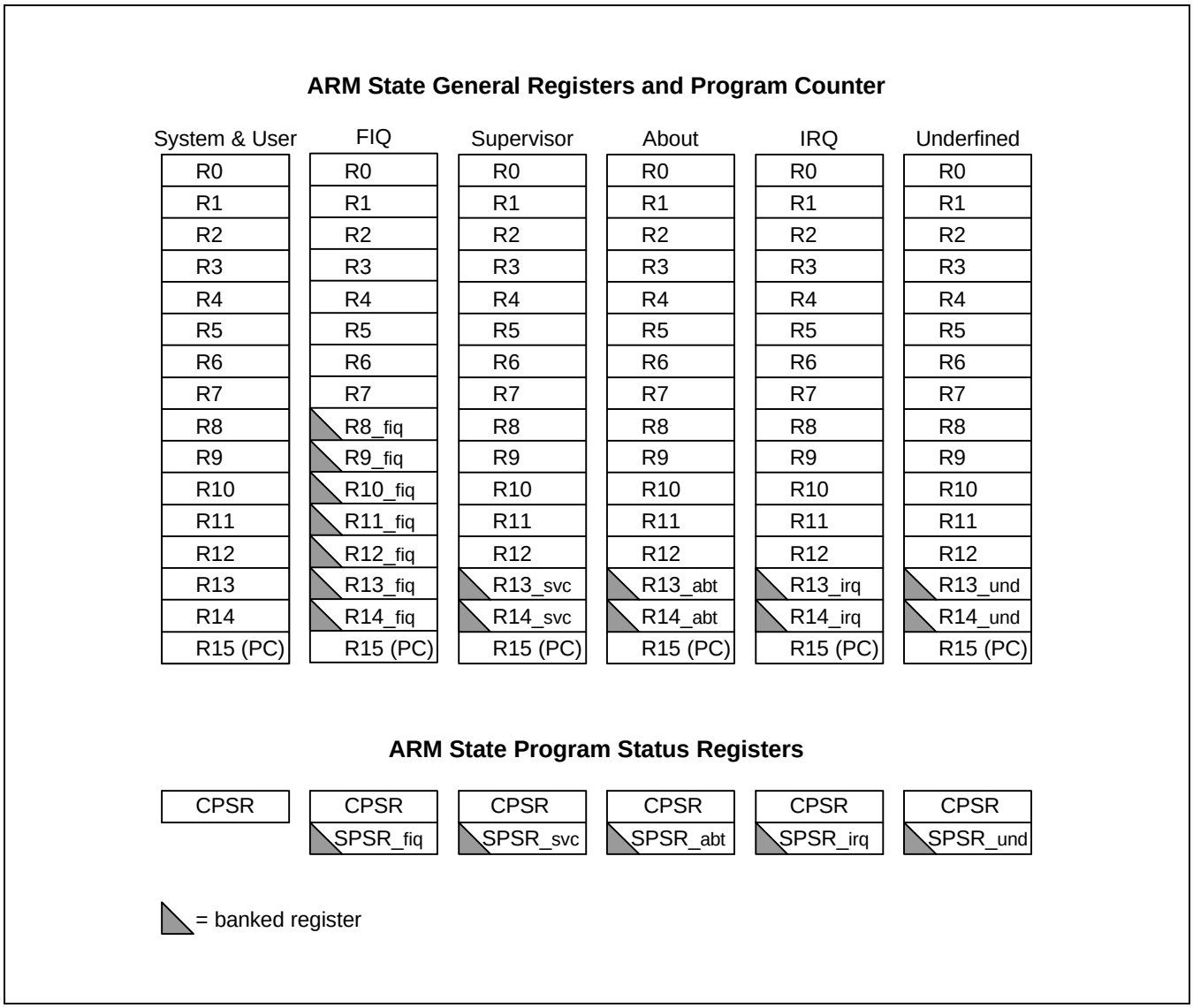


Figure 2-3. Register Organization in ARM State

The THUMB State Register Set

The THUMB state register set is a subset of the ARM state set. The programmer has direct access to eight general registers, R0-R7, as well as the Program Counter (PC), a stack pointer register (SP), a link register (LR), and the CPSR. There are banked Stack Pointers, Link Registers and Saved Process Status Registers (SPSRs) for each privileged mode. This is shown in Figure 2-4.

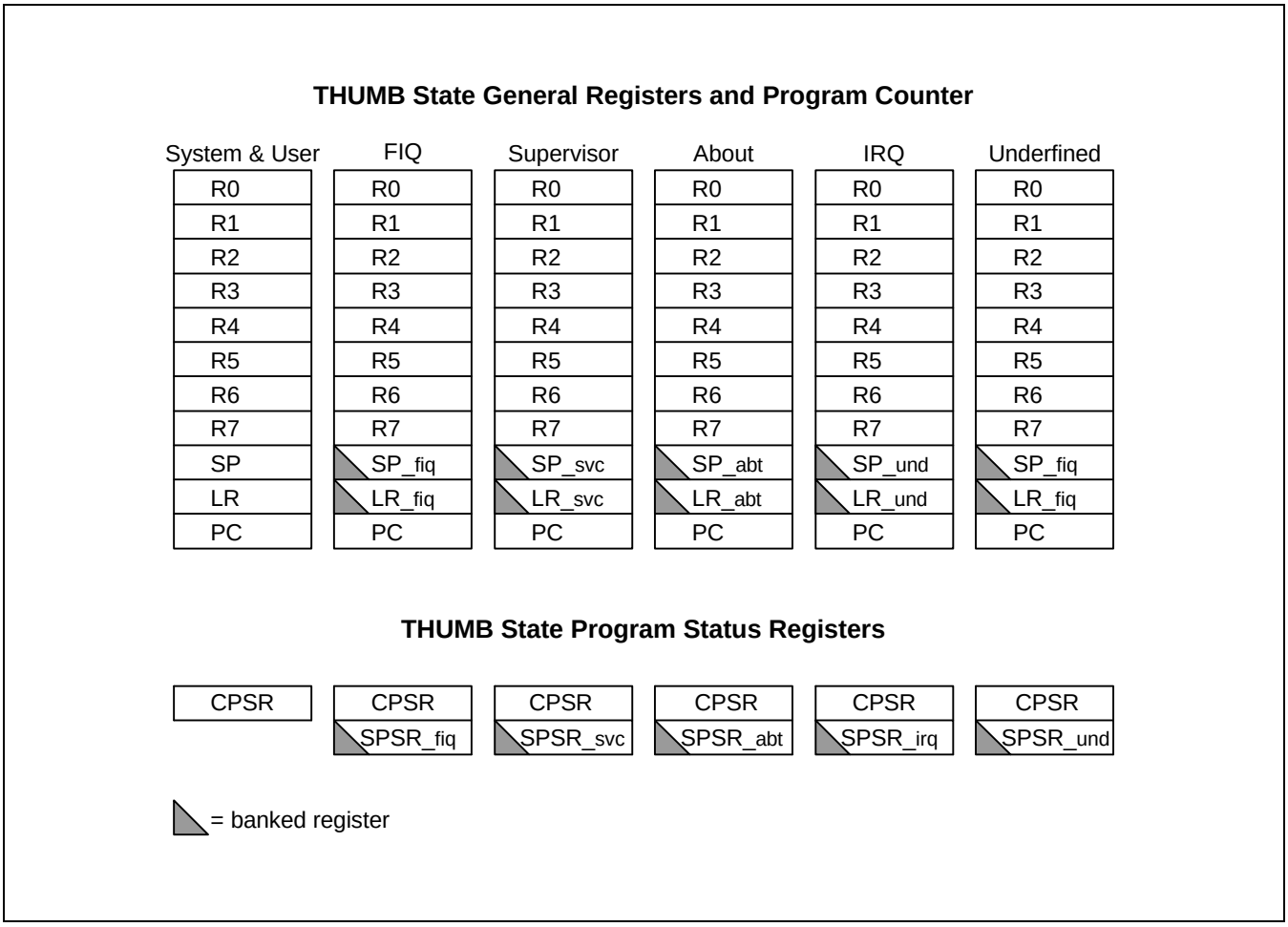


Figure 2-4. Register Organization in THUMB state

The relationship between ARM and THUMB state registers

The THUMB state registers relate to the ARM state registers in the following way:

- THUMB state R0-R7 and ARM state R0-R7 are identical
- THUMB state CPSR and SPSRs and ARM state CPSR and SPSRs are identical
- THUMB state SP maps onto ARM state R13
- THUMB state LR maps onto ARM state R14
- The THUMB state Program Counter maps onto the ARM state Program Counter (R15)

This relationship is shown in Figure 2-5.

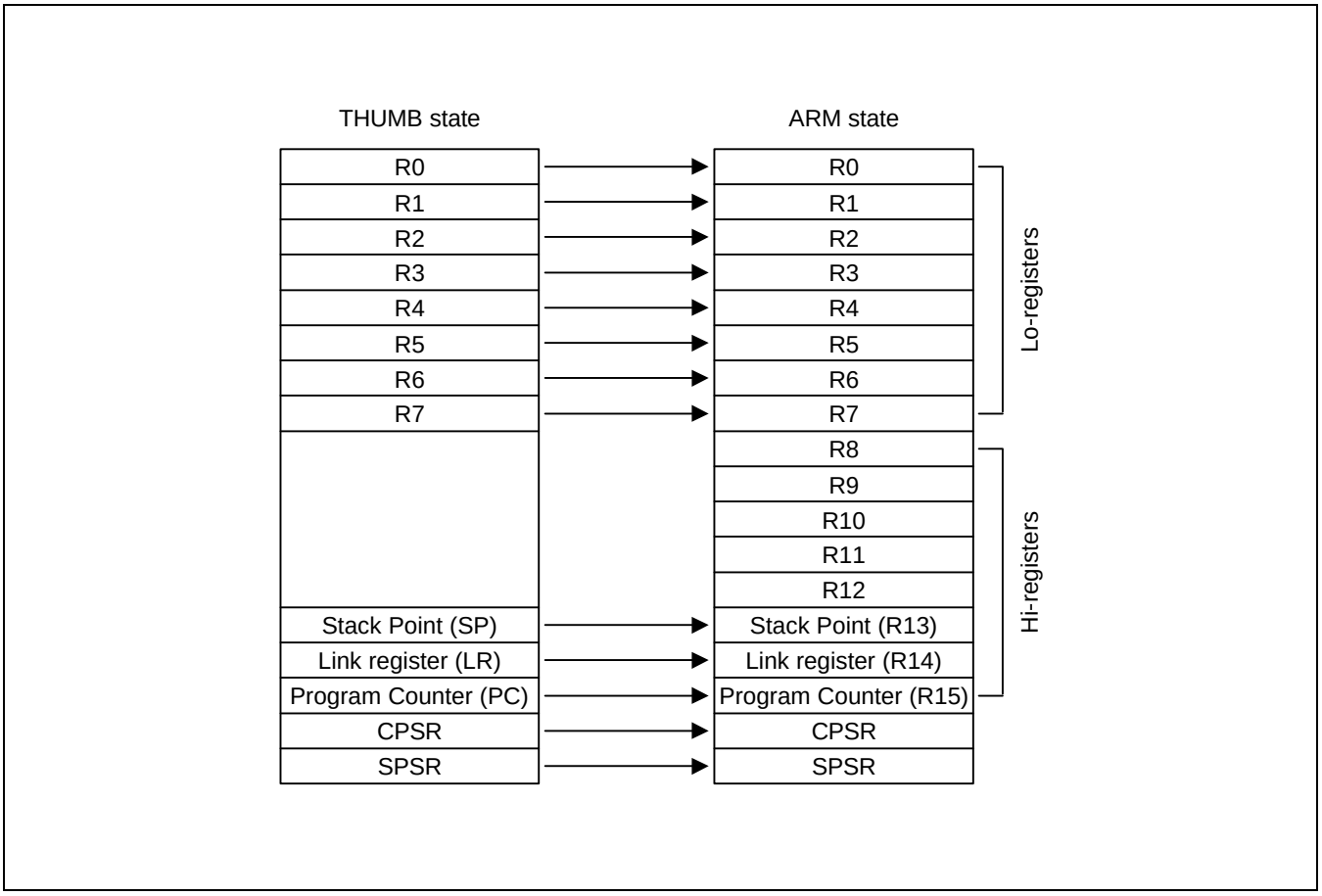


Figure 2-5. Mapping of THUMB State Registers onto ARM State Registers

Accessing Hi-Registers in THUMB State

In THUMB state, registers R8-R15 (the Hi registers) are not part of the standard register set. However, the assembly language programmer has limited access to them, and can use them for fast temporary storage.

A value may be transferred from a register in the range R0-R7 (a Lo register) to a Hi register, and from a Hi register to a Lo register, using special variants of the MOV instruction. Hi register values can also be compared against or added to Lo register values with the CMP and ADD instructions. For more information, refer to Figure 3-34.

THE PROGRAM STATUS REGISTERS

The ARM7TDMI contains a Current Program Status Register (CPSR), plus five Saved Program Status Registers (SPSRs) for use by exception handlers. These register's functions are:

- Hold information about the most recently performed ALU operation
- Control the enabling and disabling of interrupts
- Set the processor operating mode

The arrangement of bits is shown in Figure 2-6.

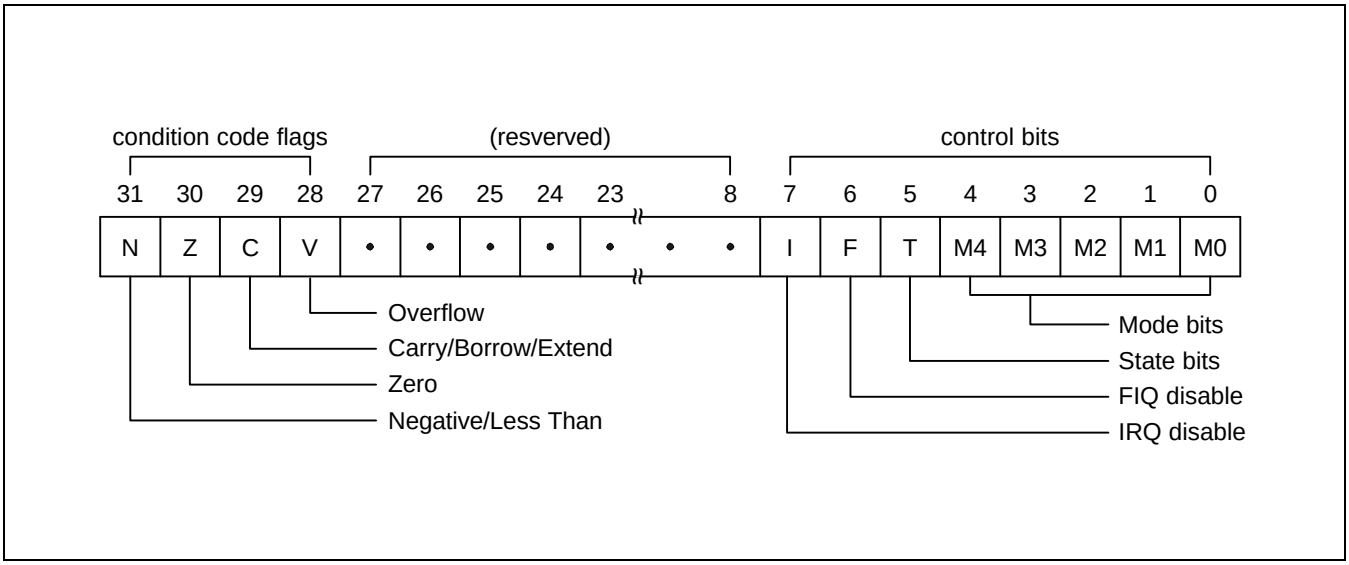


Figure 2-6. Program Status Register Format

The Condition Code Flags

The N, Z, C and V bits are the condition code flags. These may be changed as a result of arithmetic and logical operations, and may be tested to determine whether an instruction should be executed.

In ARM state, all instructions may be executed conditionally: see Table 3-2 for details.

In THUMB state, only the Branch instruction is capable of conditional execution: see Figure 3-46 for details.

The Control Bits

The bottom 8 bits of a PSR (incorporating I, F, T and M[4:0]) are known collectively as the control bits. These will change when an exception arises. If the processor is operating in a privileged mode, they can also be manipulated by software.

The T bit — This reflects the operating state. When this bit is set, the processor is executing in THUMB state, otherwise it is executing in ARM state. This is reflected on the **TBIT** external signal.

Note that the software must never change the state of the **TBIT** in the CPSR. If this happens, the processor will enter an unpredictable state.

Interrupt disable bits — The I and F bits are the interrupt disable bits. When set, these disable the IRQ and FIQ interrupts respectively.

The mode bits — The M4, M3, M2, M1 and M0 bits (M[4:0]) are the mode bits. These determine the processor's operating mode, as shown in Table 2-1. Not all combinations of the mode bits define a valid processor mode. Only those explicitly described shall be used. The user should be aware that if any illegal value is programmed into the mode bits, M[4:0], then the processor will enter an unrecoverable state. If this occurs, reset should be applied.

Table 2-1. PSR Mode Bit Values

M[4:0]	Mode	Visible THUMB state registers	Visible ARM atate registers
10000	User	R7..R0, LR, SP PC, CPSR	R14..R0, PC, CPSR
10001	FIQ	R7..R0, LR_fiq, SP_fiq PC, CPSR, SPSR_fiq	R7..R0, R14_fiq..R8_fiq, PC, CPSR, SPSR_fiq
10010	IRQ	R7..R0, LR_irq, SP_irq PC, CPSR, SPSR_irq	R12..R0, 14_irq..R13_irq, PC, CPSR, SPSR_irq
10011	Supervisor	R7..R0, LR_svc, SP_svc, PC, CPSR, SPSR_svc	R12..R0, R14_svc..R13_svc, PC, CPSR, SPSR_svc
10111	Abort	R7..R0, LR_abt, SP_abt, PC, CPSR, SPSR_abt	R12..R0, R14_abt..R13_abt, PC, CPSR, SPSR_abt
11011	Undefined	R7..R0 LR_und, SP_und, PC, CPSR, SPSR_und	R12..R0, R14_und..R13_und, PC, CPSR
11111	System	R7..R0, LR, SP PC, CPSR	R14..R0, PC, CPSR

Reserved bits — The remaining bits in the PSRs are reserved. When changing a PSR's flag or control bits, you must ensure that these unused bits are not altered. Also, your program should not rely on them containing specific values, since in future processors they may read as one or zero.

EXCEPTIONS

Exceptions arise whenever the normal flow of a program has to be halted temporarily, for example to service an interrupt from a peripheral. Before an exception can be handled, the current processor state must be preserved so that the original program can resume when the handler routine has finished.

It is possible for several exceptions to arise at the same time. If this happens, they are dealt with in a fixed order. See Exception Priorities on page 2-14.

Action on Entering an Exception

When handling an exception, the ARM7TDMI:

1. Preserves the address of the next instruction in the appropriate Link Register. If the exception has been entered from ARM state, then the address of the next instruction is copied into the Link Register (that is, current PC + 4 or PC + 8 depending on the exception. See Table 2-2 on for details). If the exception has been entered from THUMB state, then the value written into the Link Register is the current PC offset by a value such that the program resumes from the correct place on return from the exception. This means that the exception handler need not determine which state the exception was entered from. For example, in the case of SWI, MOVS PC, R14_svc will always return to the next instruction regardless of whether the SWI was executed in ARM or THUMB state.
2. Copies the CPSR into the appropriate SPSR
3. Forces the CPSR mode bits to a value which depends on the exception
4. Forces the PC to fetch the next instruction from the relevant exception vector

It may also set the interrupt disable flags to prevent otherwise unmanageable nestings of exceptions.

If the processor is in THUMB state when an exception occurs, it will automatically switch into ARM state when the PC is loaded with the exception vector address.

Action on Leaving an Exception

On completion, the exception handler:

1. Moves the Link Register, minus an offset where appropriate, to the PC. (The offset will vary depending on the type of exception.)
2. Copies the SPSR back to the CPSR
3. Clears the interrupt disable flags, if they were set on entry

NOTE

An explicit switch back to THUMB state is never needed, since restoring the CPSR from the SPSR automatically sets the T bit to the value it held immediately prior to the exception.

Exception Entry/Exit Summary

Table 2-2 summarises the PC value preserved in the relevant R14 on exception entry, and the recommended instruction for exiting the exception handler.

Table2-2. Exception Entry/Exit

	Return Instruction	Previous State		Notes
		ARM R14_x	THUMB R14_x	
BL	MOV PC, R14	PC + 4	PC + 2	1
SWI	MOVS PC, R14_svc	PC + 4	PC + 2	1
UDEF	MOVS PC, R14_und	PC + 4	PC + 2	1
FIQ	SUBS PC, R14_fiq, #4	PC + 4	PC + 4	2
IRQ	SUBS PC, R14_irq, #4	PC + 4	PC + 4	2
PABT	SUBS PC, R14_abt, #4	PC + 4	PC + 4	1
DABT	SUBS PC, R14_abt, #8	PC + 8	PC + 8	3
RESET	NA	–	–	4

NOTES:
1. Where PC is the address of the BL/SWI/Undefined Instruction fetch which had the prefetch abort.
2. Where PC is the address of the instruction which did not get executed since the FIQ or IRQ took priority.
3. Where PC is the address of the Load or Store instruction which generated the data abort.
4. The value saved in R14_svc upon reset is unpredictable.

FIQ

The FIQ (Fast Interrupt Request) exception is designed to support a data transfer or channel process, and in ARM state has sufficient private registers to remove the need for register saving (thus minimising the overhead of context switching).

FIQ is externally generated by taking the **nFIQ** input LOW. This input can except either synchronous or asynchronous transitions, depending on the state of the **ISYNC** input signal. When **ISYNC** is LOW, **nFIQ** and **nIRQ** are considered asynchronous, and a cycle delay for synchronization is incurred before the interrupt can affect the processor flow.

Irrespective of whether the exception was entered from ARM or Thumb state, a FIQ handler should leave the interrupt by executing

```
SUBS    PC,R14_fiq,#4
```

FIQ may be disabled by setting the CPSR's F flag (but note that this is not possible from User mode). If the F flag is clear, ARM7TDMI checks for a LOW level on the output of the FIQ synchroniser at the end of each instruction.

IRQ

The IRQ (Interrupt Request) exception is a normal interrupt caused by a LOW level on the **nIRQ** input. IRQ has a lower priority than FIQ and is masked out when a FIQ sequence is entered. It may be disabled at any time by setting the I bit in the CPSR, though this can only be done from a privileged (non-User) mode.

Irrespective of whether the exception was entered from ARM or Thumb state, an IRQ handler should return from the interrupt by executing

```
SUBS    PC,R14_irq,#4
```

Abort

An abort indicates that the current memory access cannot be completed. It can be signalled by the external **ABORT** input. ARM7TDMI checks for the abort exception during memory access cycles.

There are two types of abort:

- *Prefetch abort:* occurs during an instruction prefetch.
- *Data abort:* occurs during a data access.

If a prefetch abort occurs, the prefetched instruction is marked as invalid, but the exception will not be taken until the instruction reaches the head of the pipeline. If the instruction is not executed - for example because a branch occurs while it is in the pipeline - the abort does not take place.

If a data abort occurs, the action taken depends on the instruction type:

- Single data transfer instructions (LDR, STR) write back modified base registers: the Abort handler must be aware of this.
- The swap instruction (SWP) is aborted as though it had not been executed.
- Block data transfer instructions (LDM, STM) complete. If write-back is set, the base is updated. If the instruction would have overwritten the base with data (ie it has the base in the transfer list), the overwriting is prevented. All register overwriting is prevented after an abort is indicated, which means in particular that R15 (always the last register to be transferred) is preserved in an aborted LDM instruction.

The abort mechanism allows the implementation of a demand paged virtual memory system. In such a system the processor is allowed to generate arbitrary addresses. When the data at an address is unavailable, the Memory Management Unit (MMU) signals an abort. The abort handler must then work out the cause of the abort, make the requested data available, and retry the aborted instruction. The application program needs no knowledge of the amount of memory available to it, nor is its state in any way affected by the abort.

After fixing the reason for the abort, the handler should execute the following irrespective of the state (ARM or Thumb):

```
SUBS    PC,R14_abt,#4    ;  for a prefetch abort, or
SUBS    PC,R14_abt,#8    ;  for a data abort
```

This restores both the PC and the CPSR, and retries the aborted instruction.

Software Interrupt

The software interrupt instruction (SWI) is used for entering Supervisor mode, usually to request a particular supervisor function. A SWI handler should return by executing the following irrespective of the state (ARM or Thumb):

```
MOV     PC,R14_svc
```

This restores the PC and CPSR, and returns to the instruction following the SWI.

NOTE

nFIQ, nIRQ, ISYNC, LOCK, BIGEND, and ABORT pins exist only in the ARM7TDMI CPU core.

Undefined Instruction

When ARM7TDMI comes across an instruction which it cannot handle, it takes the undefined instruction trap. This mechanism may be used to extend either the THUMB or ARM instruction set by software emulation.

After emulating the failed instruction, the trap handler should execute the following irrespective of the state (ARM or Thumb):

```
MOVS    PC,R14_und
```

This restores the CPSR and returns to the instruction following the undefined instruction.

Exception Vectors

The following table shows the exception vector addresses.

Table 2-3. Exception Vectors

Address	Exception	Mode in Entry
0x00000000	Reset	Supervisor
0x00000004	Undefined instruction	Undefined
0x00000008	Software Interrupt	Supervisor
0x0000000C	Abort (prefetch)	Abort
0x00000010	Abort (data)	Abort
0x00000014	Reserved	Reserved
0x00000018	IRQ	IRQ
0x0000001C	FIQ	FIQ

Exception Priorites

When multiple exceptions arise at the same time, a fixed priority system determines the order in which they are handled:

Highest priority:

1. Reset
2. Data abort
3. FIQ
4. IRQ
5. Prefetch abort

Lowest priority:

6. Undefined Instruction, Software interrupt.

Not All Exceptions Can Occur at Once:

Undefined Instruction and Software Interrupt are mutually exclusive, since they each correspond to particular (non-overlapping) decodings of the current instruction.

If a data abort occurs at the same time as a FIQ, and FIQs are enabled (ie the CPSR's F flag is clear), ARM7TDMI enters the data abort handler and then immediately proceeds to the FIQ vector. A normal return from FIQ will cause the data abort handler to resume execution. Placing data abort at a higher priority than FIQ is necessary to ensure that the transfer error does not escape detection. The time for this exception entry should be added to worst-case FIQ latency calculations.

INTERRUPT LATENCIES

The worst case latency for FIQ, assuming that it is enabled, consists of the longest time the request can take to pass through the synchroniser (*Tsyncmax* if asynchronous), plus the time for the longest instruction to complete (*Tldm*, the longest instruction is an LDM which loads all the registers including the PC), plus the time for the data abort entry (*Texc*), plus the time for FIQ entry (*Tfiq*). At the end of this time ARM7TDMI will be executing the instruction at 0x1C.

Tsyncmax is 3 processor cycles, *Tldm* is 20 cycles, *Texc* is 3 cycles, and *Tfiq* is 2 cycles. The total time is therefore 28 processor cycles. This is just over 1.4 microseconds in a system which uses a continuous 20 MHz processor clock. The maximum IRQ latency calculation is similar, but must allow for the fact that FIQ has higher priority and could delay entry into the IRQ handling routine for an arbitrary length of time. The minimum latency for FIQ or IRQ consists of the shortest time the request can take through the synchroniser (*Tsyncmin*) plus *Tfiq*. This is 4 processor cycles.

RESET

When the **nRESET** signal goes LOW, ARM7TDMI abandons the executing instruction and then continues to fetch instructions from incrementing word addresses.

When **nRESET** goes HIGH again, ARM7TDMI:

1. Overwrites R14_svc and SPSR_svc by copying the current values of the PC and CPSR into them. The value of the saved PC and SPSR is not defined.
2. Forces M[4:0] to 10011 (Supervisor mode), sets the I and F bits in the CPSR, and clears the CPSR's T bit.
3. Forces the PC to fetch the next instruction from address 0x00.
4. Execution resumes in ARM state.

NOTES

3 INSTRUCTION SET

INSTRUCTION SET SUMMAY

This chapter describes the ARM instruction set and the THUMB instruction set in the ARM7TDMI core.

FORMAT SUMMARY

The ARM instruction set formats are shown below.

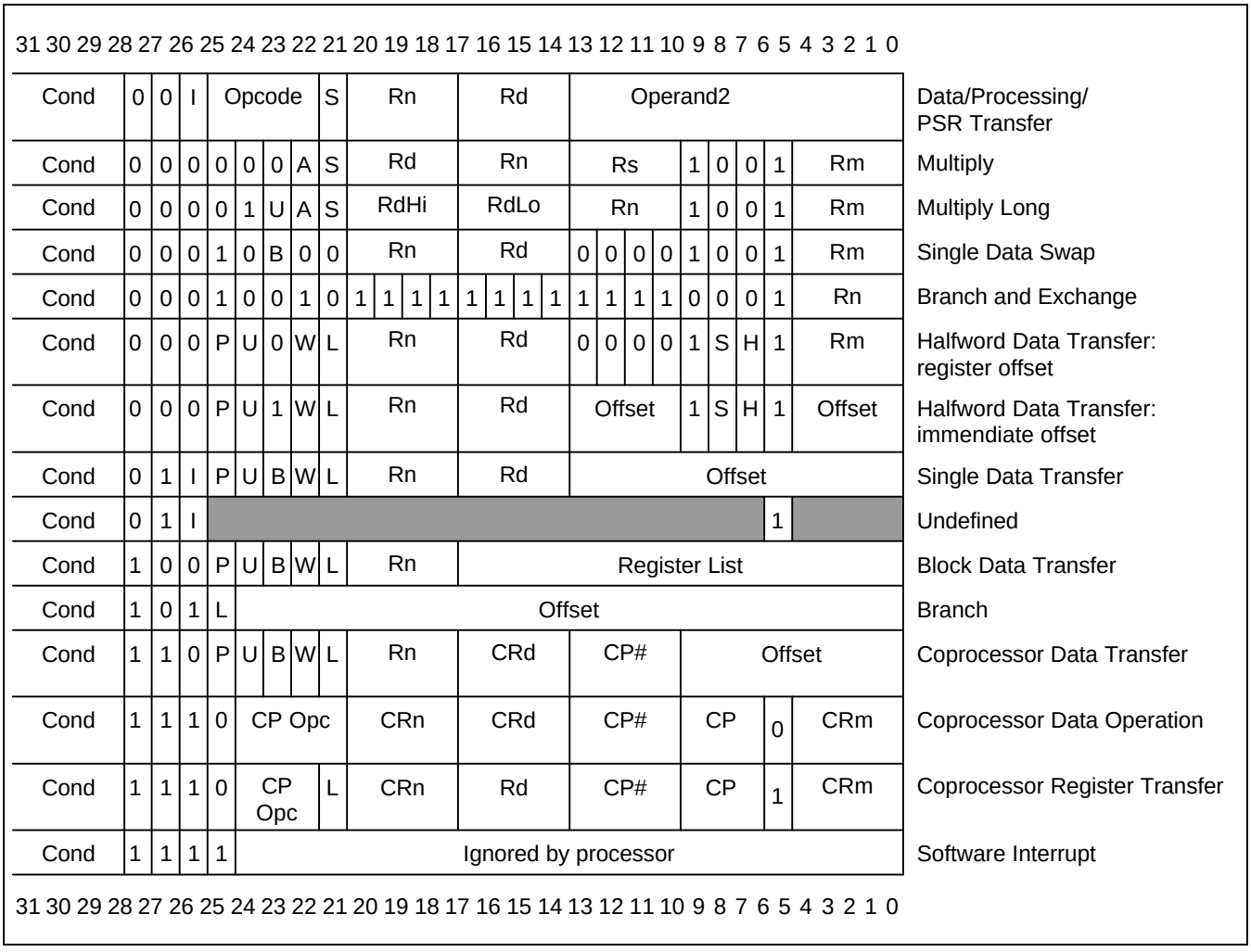


Figure 3-1. ARM Instruction Set Format

NOTE

Some instruction codes are not defined but do not cause the Undefined instruction trap to be taken, for instance a Multiply instruction with bit 6 changed to a 1. These instructions should not be used, as their action may change in future ARM implementations.

INSTRUCTION SUMMARY

Table 3-1. The ARM Instruction Set

Mnemonic	Instruction	Action
ADC	Add with carry	Rd: = Rn + Op2 + Carry
ADD	Add	Rd: = Rn + Op2
AND	ANDo	Rd: = Rn AND Op2
B	Branch	R15: = address
BIC	Bit Clear	Rd: = Rn AND NOT Op2
BL	Branch with Link	R14: = R15, R15: = address
BX	Branch and Exchange	R15: = Rn, T bit: = Rn[0]
CDP	Coprocessor Data Processing	(Coprocessor-specific)
CMN	Compare Negative	CPSR flags: = Rn + Op2
CMP	Compare	CPSR flags: = Rn - Op2
EOR	Exclusive OR	Rd: = (Rn AND NOT Op2) OR (Op2 AND NOT Rn)
LDC	Load coprocessor from memory	Coprocessor load
LDM	Load multiple registers	Stack manipulation (Pop)
LDR	Load register from memory	Rd: = (address)
MCR	Move CPU register to coprocessor register	cRn: = rRn {<op>cRm}
MLA	Multiply Accumulate	Rd: = (Rm × Rs) + Rn
MOV	Move register or constant	Rd: = Op2
MRC	Move from coprocessor register to CPU register	Rn: = cRn {<op>cRm}
MRS	Move PSR status/flags to register	Rn: = PSR
MSR	Move register to PSR status/flags	PSR: = Rm
MUL	Multiply	Rd: = Rm × Rs

Table 3-1. The ARM Instruction Set (Continued)

Mnemonic	Instruction	Action
MVN	Move negative register	Rd: = 0 × FFFFFFFF EOR Op2
ORR	OR	Rd: = Rn OR Op2
RSB	Reverse Subtract	Rd: = Op2 - Rn
RSC	Reverse Subtract with Carry	Rd: = Op2 - Rn - 1 + Carry
SBC	Subtract with Carry	Rd: = Rn - Op2 - 1 + Carry
STC	Store coprocessor register to memory	address: = CRn
STM	Store Multiple	Stack manipulation (Push)
STR	Store register to memory	<address>: = Rd
SUB	Subtract	Rd: = Rn - Op2
SWI	Software Interrupt	OS call
SWP	Swap register with memory	Rd: = [Rn], [Rn] := Rm
TEQ	Test bitwise equality	CPSR flags: = Rn EOR Op2
TST	Test bits	CPSR flags: = Rn AND Op2

THE CONDITION FIELD

In ARM state, all instructions are conditionally executed according to the state of the CPSR condition codes and the instruction's condition field. This field (bits 31:28) determines the circumstances under which an instruction is to be executed. If the state of the C, N, Z and V flags fulfils the conditions encoded by the field, the instruction is executed, otherwise it is ignored.

There are sixteen possible conditions, each represented by a two-character suffix that can be appended to the instruction's mnemonic. For example, a Branch (B in assembly language) becomes BEQ for "Branch if Equal", which means the Branch will only be taken if the Z flag is set.

In practice, fifteen different conditions may be used: these are listed in Table 3-2. The sixteenth (1111) is reserved, and must not be used.

In the absence of a suffix, the condition field of most instructions is set to "Always" (sufix AL). This means the instruction will always be executed regardless of the CPSR condition codes.

Table3-2. Condition Code Summary

Code	Suffix	Flags	Meaning
0000	EQ	Z set	equal
0001	NE	Z clear	not equal
0010	CS	C set	unsigned higher or same
0011	CC	C clear	unsigned lower
0100	MI	N set	negative
0101	PL	N clear	positive or zero
0110	VS	V set	overflow
0111	VC	V clear	no overflow
1000	HI	C set and Z clear	unsigned higher
1001	LS	C clear or Z set	unsigned lower or same
1010	GE	N equals V	greater or equal
1011	LT	N not equal to V	less than
1100	GT	Z clear AND (N equals V)	greater than
1101	LE	Z set OR (N not equal to V)	less than or equal
1110	AL	(ignored)	always

BRANCH AND EXCHANGE (BX)

This instruction is only executed if the condition is true. The various conditions are defined in Table 3-2.

This instruction performs a branch by copying the contents of a general register, Rn, into the program counter, PC. The branch causes a pipeline flush and refill from the address specified by Rn. This instruction also permits the instruction set to be exchanged. When the instruction is executed, the value of Rn[0] determines whether the instruction stream will be decoded as ARM or THUMB instructions.

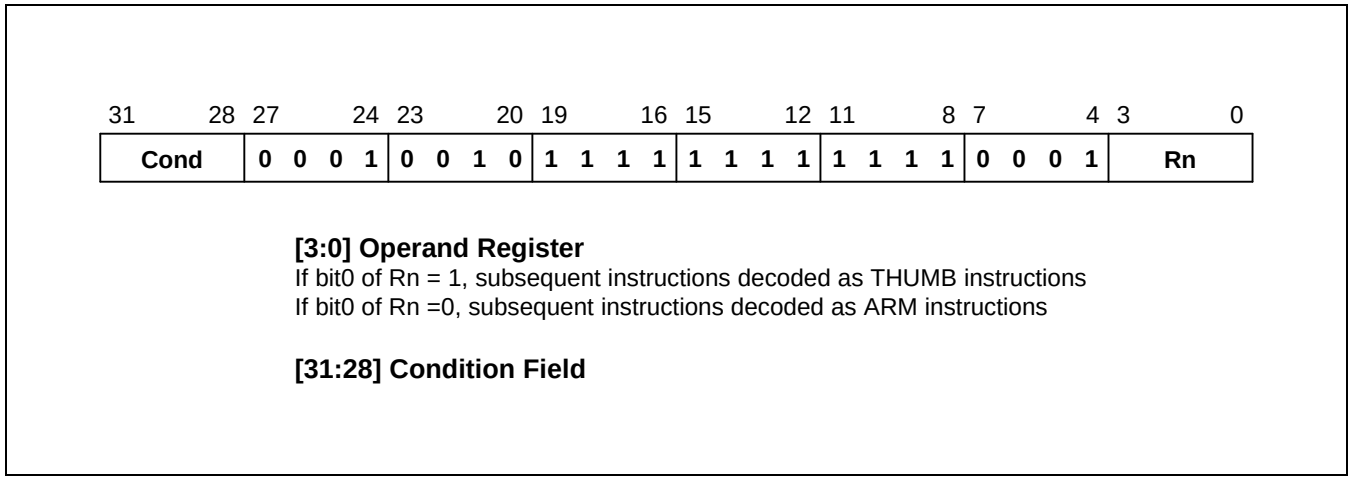


Figure 3-2. Branch and Exchange Instructions

INSTRUCTION CYCLE TIMES

The BX instruction takes 2S + 1N cycles to execute, where S and N are defined as sequential (S-cycle) and non-sequencial (N-cycle), respectively.

ASSEMBLER SYNTAX

BX - branch and exchange.

BX {cond} Rn

{cond}	Two character condition mnemonic. See Table 3-2.
Rn	is an expression evaluating to a valid register number.

USING R15 AS AN OPERAND

If R15 is used as an operand, the behaviour is undefined.

Examples

```
ADR       R0, Into_THUMB + 1    ; Generate branch target address
                                ; and set bit 0 high - hence
                                ; arrive in THUMB state.
BX        R0                    ; Branch and change to THUMB
                                ; state.
CODE16                          ; Assemble subsequent code as
Into_THUMB                      ; THUMB instructions
•
•
•
ADR R5, Back_to_ARM             ; Generate branch target to word aligned address
                                ; - hence bit 0 is low and so change back to ARM state.
BX R5                           ; Branch and change back to ARM state.
•
•
•
ALIGN                           ; Word align
CODE32                          ; Assemble subsequent code as ARM instructions
Back_to_ARM
```

BRANCH AND BRANCH WITH LINK (B, BL)

The instruction is only executed if the condition is true. The various conditions are defined Table 3-2. The instruction encoding is shown in Figure 3-3, below.

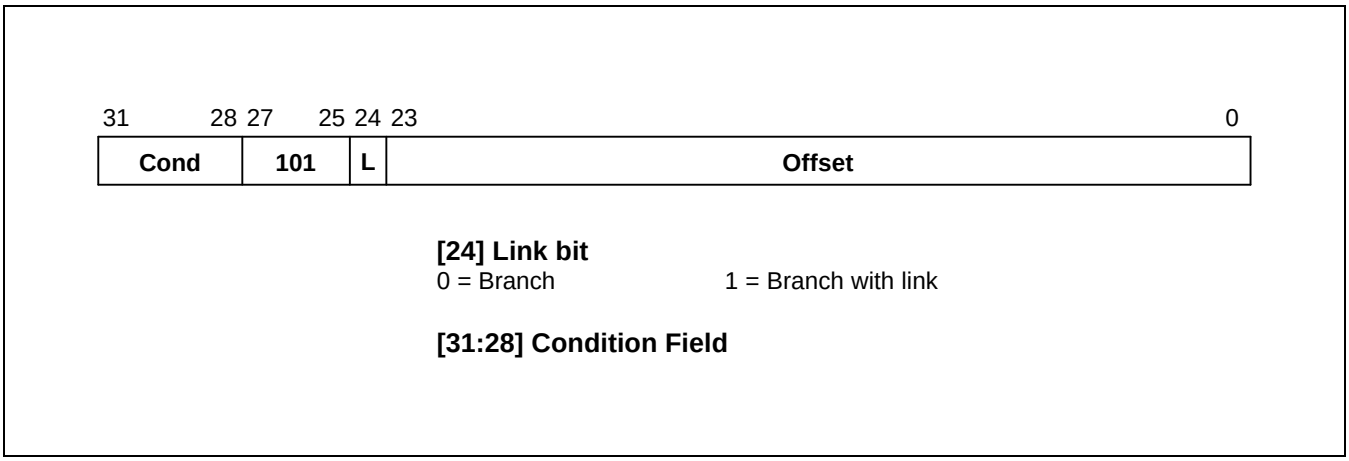


Figure 3-3. Branch Instructions

Branch instructions contain a signed 2's complement 24 bit offset. This is shifted left two bits, sign extended to 32 bits, and added to the PC. The instruction can therefore specify a branch of +/- 32Mbytes. The branch offset must take account of the prefetch operation, which causes the PC to be 2 words (8 bytes) ahead of the current instruction.

Branches beyond +/- 32Mbytes must use an offset or absolute destination which has been previously loaded into a register. In this case the PC should be manually saved in R14 if a Branch with Link type operation is required.

THE LINK BIT

Branch with Link (BL) writes the old PC into the link register (R14) of the current bank. The PC value written into R14 is adjusted to allow for the prefetch, and contains the address of the instruction following the branch and link instruction. Note that the CPSR is not saved with the PC and R14[1:0] are always cleared.

To return from a routine called by Branch with Link use MOV PC,R14 if the link register is still valid or LDM Rn!,{..PC} if the link register has been saved onto a stack pointed to by Rn.

INSTRUCTION CYCLE TIMES

Branch and Branch with Link instructions take 2S + 1N incremental cycles, where S and N are defined as squential (S-cycle) and internal (I-cycle).

ASSEMBLER SYNTAX

Items in {} are optional. Items in <> must be present.

B{L}{cond} <expression>

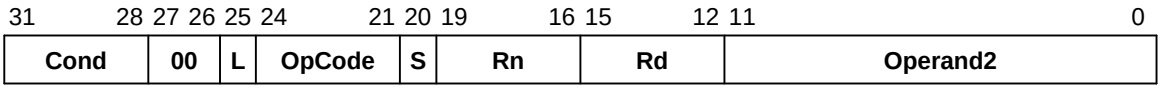
{L}	Used to request the Branch with Link form of the instruction. If absent, R14 will not be affected by the instruction.
{cond}	A two-character mnemonic as shown in Table 3-2. If absent then AL (ALways) will be used.
<expression>	The destination. The assembler calculates the offset.

EXAMPLES

```
here    BAL     here        ;  Assembles to 0xEAFFFFFE (note effect of PC offset).
        B       there       ;  Always condition used as default.
        CMP     R1,#0       ;  Compare R1 with zero and branch to fred
                            ;  if R1 was zero, otherwise continue.
        BEQ     fred        ;  Continue to next instruction.
        BL      sub+ROM     ;  Call subroutine at computed address.
        ADDS    R1,#1       ;  Add 1 to register 1, setting CPSR flags
                            ;  on the result then call subroutine if
        BLCC    sub         ;  the C flag is clear, which will be the
                            ;  case unless R1 held 0xFFFFFFFF.
```

DATA PROCESSING

The data processing instruction is only executed if the condition is true. The conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-4.



[15:12] Destination register
0 = Branch 1 = Branch with Link

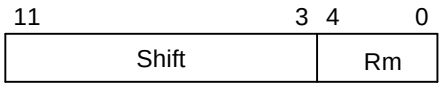
[19:16] 1st operand register
0 = Branch 1 = Branch with Link

[20] Set condition codes
0 = Do not after condition codes 1 = Set condition codes

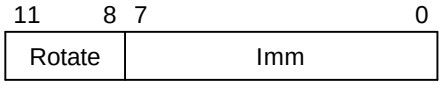
[24:21] Operation code
0000 = AND-Rd: = Op1 AND Op2
0001 = EOR-Rd: = Op1 EOR Op2
0010 = SUB-Rd: = Op1-Op2
0011 = RSB-Rd: = Op2-Op1
0100 = ADD-Rd: = Op1+Op2
0101 = ADC-Rd: = Op1+Op2+C
0110 = SBC-Rd: = OP1-Op2+C-1
0111 = RSC-Rd: = Op2-Op1+C-1
1000 = TST-set condition codes on Op1 AND Op2
1001 = TEO-set condition codes on OP1 EOR Op2
1010 = CMP-set condition codes on Op1-Op2
1011 = SMN-set condition codes on Op1+Op2
1100 = ORR-Rd: = Op1 OR Op2
1101 = MOV-Rd: =OP2
1110 = BIC-Rd: = Op1 AND NOT Op2
1111 = MVN-Rd: = NOT Op2

[25] Immediate operand
0 = Operand 2 is a register 1 = Operand 2 is an immediate Value

[11:0] Operand 2 Type selection



[3:0] 2nd Operand Register [11:4] Shift applied to Rm



[7:0] Unsigned 8 bit immediate value [11:8] Shift applied to Imm

[31:28] Condition field

Figure 3-4. Data Processing Instructions

The instruction produces a result by performing a specified arithmetic or logical operation on one or two operands. The first operand is always a register (Rn).

The second operand may be a shifted register (Rm) or a rotated 8 bit immediate value (Imm) according to the value of the I bit in the instruction. The condition codes in the CPSR may be preserved or updated as a result of this instruction, according to the value of the S bit in the instruction.

Certain operations (TST, TEQ, CMP, CMN) do not write the result to Rd. They are used only to perform tests and to set the condition codes on the result and always have the S bit set. The instructions and their effects are listed in Table 3-3.

CPSR FLAGS

The data processing operations may be classified as logical or arithmetic. The logical operations (AND, EOR, TST, TEQ, ORR, MOV, BIC, MVN) perform the logical action on all corresponding bits of the operand or operands to produce the result. If the S bit is set (and Rd is not R15, see below) the V flag in the CPSR will be unaffected, the C flag will be set to the carry out from the barrel shifter (or preserved when the shift operation is LSL #0), the Z flag will be set if and only if the result is all zeros, and the N flag will be set to the logical value of bit 31 of the result.

Table 3-3. ARM Data Processing Instructions

Assembler Mnemonic	OP Code	Action
AND	0000	Operand1 AND operand2
EOR	0001	Operand1 EOR operand2
WUB	0010	Operand1 - operand2
RSB	0011	Operand2 operand1
ADD	0100	Operand1 + operand2
ADC	0101	Operand1 + operand2 + carry
SBC	0110	Operand1 - operand2 + carry - 1
RSC	0111	Operand2 - operand1 + carry - 1
TST	1000	As AND, but result is not written
TEQ	1001	As EOR, but result is not written
CMP	1010	As SUB, but result is not written
CMN	1011	As ADD, but result is not written
ORR	1100	Operand1 OR operand2
MOV	1101	Operand2 (operand1 is ignored)
BIC	1110	Operand1 AND NOT operand2 (Bit clear)
MVN	1111	NOT operand2 (operand1 is ignored)

The arithmetic operations (SUB, RSB, ADD, ADC, SBC, RSC, CMP, CMN) treat each operand as a 32 bit integer (either unsigned or 2's complement signed, the two are equivalent). If the S bit is set (and Rd is not R15) the V flag in the CPSR will be set if an overflow occurs into bit 31 of the result; this may be ignored if the operands were considered unsigned, but warns of a possible error if the operands were 2's complement signed. The C flag will be set to the carry out of bit 31 of the ALU, the Z flag will be set if and only if the result was zero, and the N flag will be set to the value of bit 31 of the result (indicating a negative result if the operands are considered to be 2's complement signed).

SHIFTS

When the second operand is specified to be a shifted register, the operation of the barrel shifter is controlled by the Shift field in the instruction. This field indicates the type of shift to be performed (logical left or right, arithmetic right or rotate right). The amount by which the register should be shifted may be contained in an immediate field in the instruction, or in the bottom byte of another register (other than R15). The encoding for the different shift types is shown in Figure 3-5.

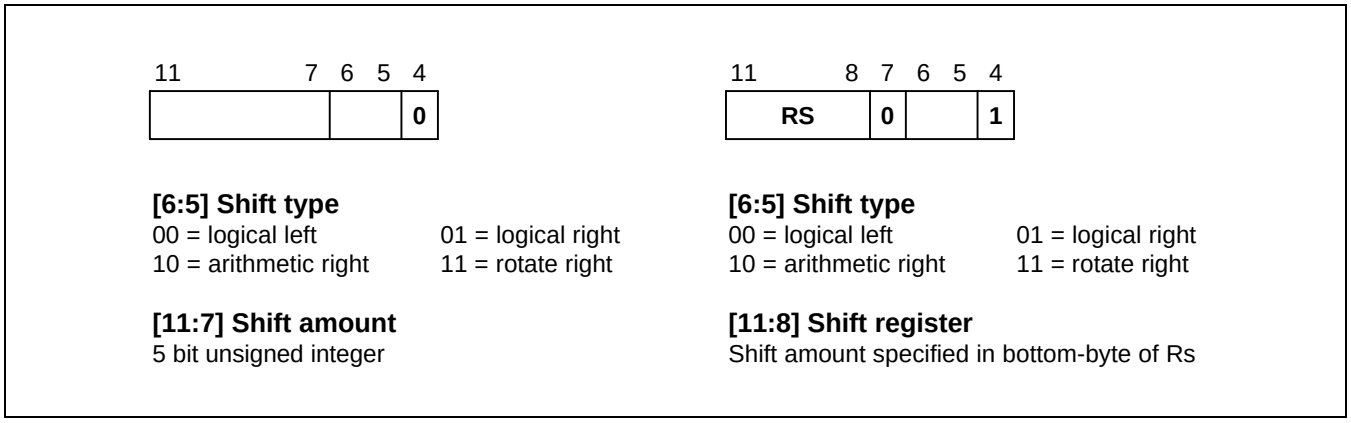


Figure 3-5. ARM Shift Operations

Instruction specified shift amount

When the shift amount is specified in the instruction, it is contained in a 5 bit field which may take any value from 0 to 31. A logical shift left (LSL) takes the contents of Rm and moves each bit by the specified amount to a more significant position. The least significant bits of the result are filled with zeros, and the high bits of Rm which do not map into the result are discarded, except that the least significant discarded bit becomes the shifter carry output which may be latched into the C bit of the CPSR when the ALU operation is in the logical class (see above). For example, the effect of LSL #5 is shown in Figure 3-6.

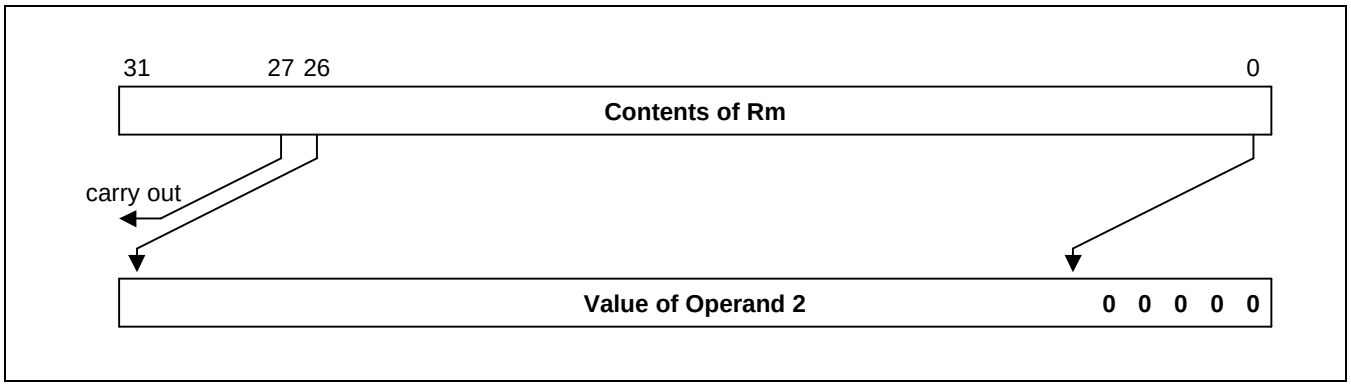


Figure 3-6. Logical Shift Left

NOTE

LSL #0 is a special case, where the shifter carry out is the old value of the CPSR C flag. The contents of Rm are used directly as the second operand. A logical shift right (LSR) is similar, but the contents of Rm are moved to less significant positions in the result. LSR #5 has the effect shown in Figure 3-7.

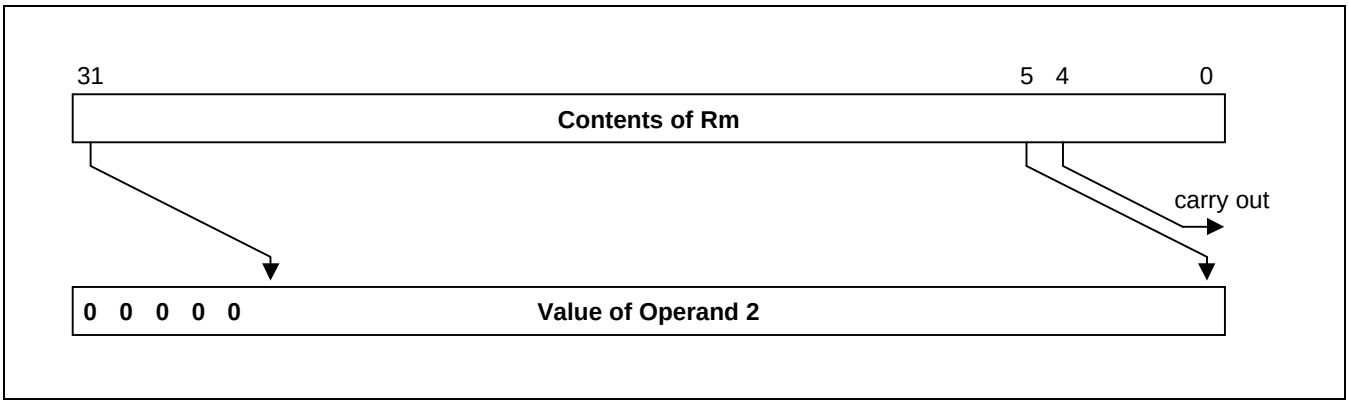


Figure 3-7. Logical Shift Right

The form of the shift field which might be expected to correspond to LSR #0 is used to encode LSR #32, which has a zero result with bit 31 of Rm as the carry output. Logical shift right zero is redundant as it is the same as logical shift left zero, so the assembler will convert LSR #0 (and ASR #0 and ROR #0) into LSL #0, and allow LSR #32 to be specified.

An arithmetic shift right (ASR) is similar to logical shift right, except that the high bits are filled with bit 31 of Rm instead of zeros. This preserves the sign in 2's complement notation. For example, ASR #5 is shown in Figure 3-8.

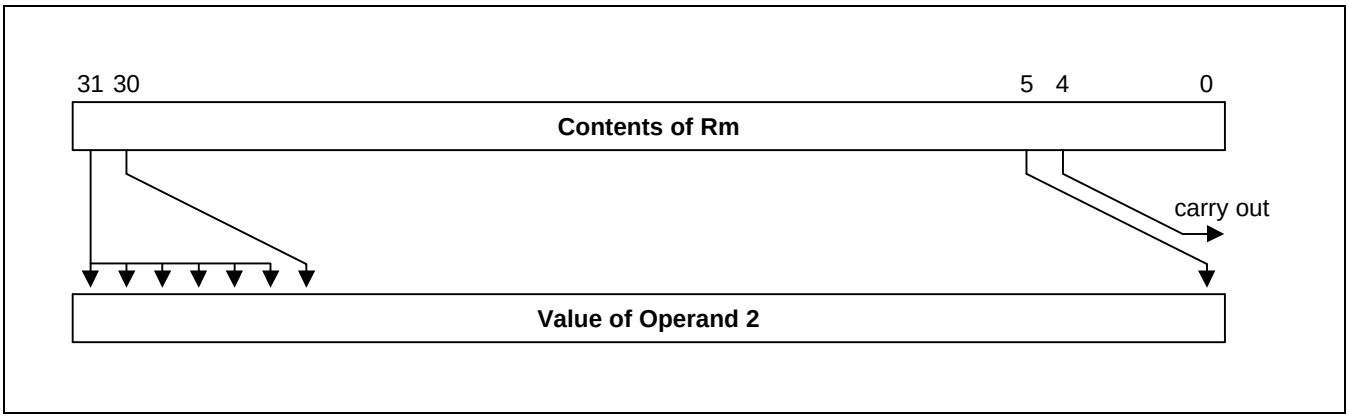


Figure 3-8. Arithmetic Shift Right

The form of the shift field which might be expected to give ASR #0 is used to encode ASR #32. Bit 31 of Rm is again used as the carry output, and each bit of operand 2 is also equal to bit 31 of Rm. The result is therefore all ones or all zeros, according to the value of bit 31 of Rm.

Rotate right (ROR) operations reuse the bits which "overshoot" in a logical shift right operation by reintroducing them at the high end of the result, in place of the zeros used to fill the high end in logical right operations. For example, ROR #5 is shown in Figure 3-9.

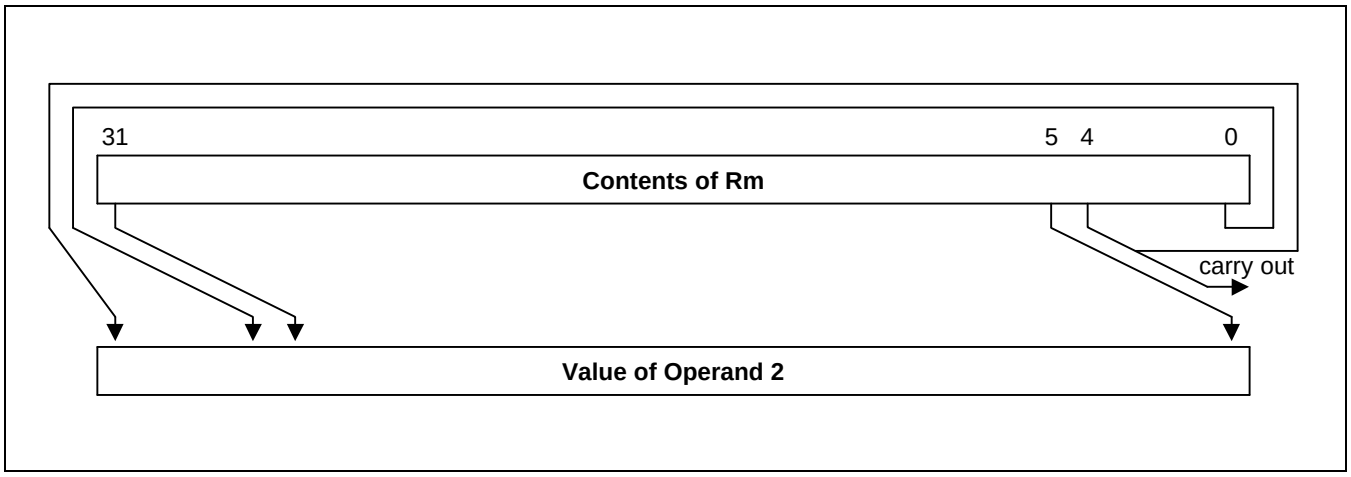


Figure 3-9. Rotate Right

The form of the shift field which might be expected to give ROR #0 is used to encode a special function of the barrel shifter, rotate right extended (RRX). This is a rotate right by one bit position of the 33 bit quantity formed by appending the CPSR C flag to the most significant end of the contents of Rm as shown in Figure 3-10.

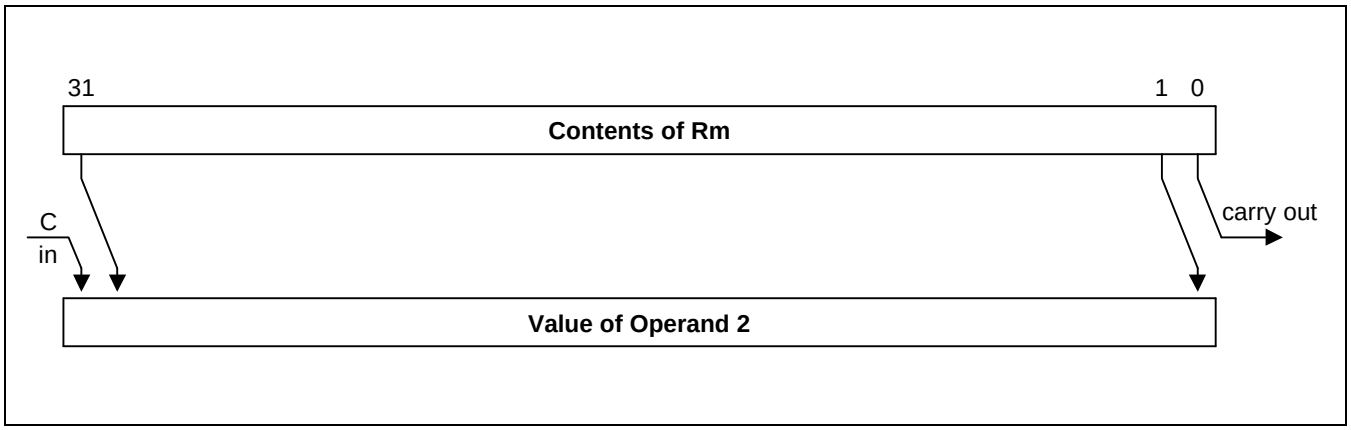


Figure 3-10. Rotate Right Extended

Register specified shift amount

Only the least significant byte of the contents of Rs is used to determine the shift amount. Rs can be any general register other than R15.

If this byte is zero, the unchanged contents of Rm will be used as the second operand, and the old value of the CPSR C flag will be passed on as the shifter carry output.

If the byte has a value between 1 and 31, the shifted result will exactly match that of an instruction specified shift with the same value and shift operation.

If the value in the byte is 32 or more, the result will be a logical extension of the shift described above:

1. LSL by 32 has result zero, carry out equal to bit 0 of Rm.
2. LSL by more than 32 has result zero, carry out zero.
3. LSR by 32 has result zero, carry out equal to bit 31 of Rm.
4. LSR by more than 32 has result zero, carry out zero.
5. ASR by 32 or more has result filled with and carry out equal to bit 31 of Rm.
6. ROR by 32 has result equal to Rm, carry out equal to bit 31 of Rm.
7. ROR by n where n is greater than 32 will give the same result and carry out as ROR by n-32; therefore repeatedly subtract 32 from n until the amount is in the range 1 to 32 and see above.

NOTE

The zero in bit 7 of an instruction with a register controlled shift is compulsory; a one in this bit will cause the instruction to be a multiply or undefined instruction.

IMMEDIATE OPERAND ROTATES

The immediate operand rotate field is a 4 bit unsigned integer which specifies a shift operation on the 8 bit immediate value. This value is zero extended to 32 bits, and then subject to a rotate right by twice the value in the rotate field. This enables many common constants to be generated, for example all powers of 2.

WRITING TO R15

When Rd is a register other than R15, the condition code flags in the CPSR may be updated from the ALU flags as described above.

When Rd is R15 and the S flag in the instruction is not set the result of the operation is placed in R15 and the CPSR is unaffected.

When Rd is R15 and the S flag is set the result of the operation is placed in R15 and the SPSR corresponding to the current mode is moved to the CPSR. This allows state changes which atomically restore both PC and CPSR. This form of instruction should not be used in User mode.

USING R15 AS AN OPERAND

If R15 (the PC) is used as an operand in a data processing instruction the register is used directly.

The PC value will be the address of the instruction, plus 8 or 12 bytes due to instruction prefetching. If the shift amount is specified in the instruction, the PC will be 8 bytes ahead. If a register is used to specify the shift amount the PC will be 12 bytes ahead.

TEQ, TST, CMP AND CMN OPCODES

NOTE

TEQ, TST, CMP and CMN do not write the result of their operation but do set flags in the CPSR. An assembler should always set the S flag for these instructions even if this is not specified in the mnemonic.

The TEQP form of the TEQ instruction used in earlier ARM processors must not be used: the PSR transfer operations should be used instead.

The action of TEQP in the ARM7TDMI is to move SPSR_<mode> to the CPSR if the processor is in a privileged mode and to do nothing if in User mode.

INSTRUCTION CYCLE TIMES

Data Processing instructions vary in the number of incremental cycles taken as follows:

Table 3-4. Incremental Cycle Times

Processing Type	Cycles
Normal data processing	1S
Data processing with register specified shift	1S + 1I
Data processing with PC written	2S + 1N
Data processing with register specified shift and PC written	2S + 1N +1I

NOTE: S, N and I are as defined sequential (S-cycle), non-sequential (N-cycle), and internal (I-cycle) respectively.

ASSEMBLER SYNTAX

- MOV,MVN (single operand instructions).
  <opcode>{cond}{S} Rd,<Op2>
- CMP,CMN,TEQ,TST (instructions which do not produce a result).
  <opcode>{cond} Rn,<Op2>
- AND,EOR,SUB,RSB,ADD,ADC,SBC,RSC,ORR,BIC
  <opcode>{cond}{S} Rd,Rn,<Op2>

where:

<Op2>	Rm{,<shift>} or,<#expression>
{cond}	A two-character condition mnemonic. See Table 3-2.
{S}	Set condition codes if S present (implied for CMP, CMN, TEQ, TST).
Rd, Rn and Rm	Expressions evaluating to a register number.
<#expression>	If this is used, the assembler will attempt to generate a shifted immediate 8-bit field to match the expression. If this is impossible, it will give an error.
<shift>	<Shiftname> <register> or <shiftname> #expression, or RRX (rotate right one bit with extend).
<shiftname>s	ASL, LSL, LSR, ASR, ROR. (ASL is a synonym for LSL, they assemble to the same code.)

EXAMPLES

```
ADDEQ   R2,R4,R5        ; If the Z flag is set make R2:=R4+R5
TEQS    R4,#3           ; Test R4 for equality with 3.
                        ; (The S is in fact redundant as the
                        ; assembler inserts it automatically.)
SUB     R4,R5,R7,LSR R2 ; Logical right shift R7 by the number in
                        ; the bottom byte of R2, subtract result
                        ; from R5, and put the answer into R4.
MOV     PC,R14          ; Return from subroutine.
MOVS    PC,R14          ; Return from exception and restore CPSR
                        ; from SPSR_mode.
```

PSR TRANSFER (MRS, MSR)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2.

The MRS and MSR instructions are formed from a subset of the Data Processing operations and are implemented using the TEQ, TST, CMN and CMP instructions without the S flag set. The encoding is shown in Figure 3-11.

These instructions allow access to the CPSR and SPSR registers. The MRS instruction allows the contents of the CPSR or SPSR_<mode> to be moved to a general register. The MSR instruction allows the contents of a general register to be moved to the CPSR or SPSR_<mode> register.

The MSR instruction also allows an immediate value or register contents to be transferred to the condition code flags (N,Z,C and V) of CPSR or SPSR_<mode> without affecting the control bits. In this case, the top four bits of the specified register contents or 32 bit immediate value are written to the top four bits of the relevant PSR.

OPERAND RESTRICTIONS

- In user mode, the control bits of the CPSR are protected from change, so only the condition code flags of the CPSR can be changed. In other (privileged) modes the entire CPSR can be changed.
- Note that the software must never change the state of the T bit in the CPSR. If this happens, the processor will enter an unpredictable state.
- The SPSR register which is accessed depends on the mode at the time of execution. For example, only SPSR_fiq is accessible when the processor is in FIQ mode.
- You must not specify R15 as the source or destination register.
- Also, do not attempt to access an SPSR in User mode, since no such register exists.

MRS (transfer PSR contents to a register)

31 28	27 23	22	21 16	15 12	11 0
Cond	00010	Ps	001111	Rd	000000000000

[15:21] Destination Register

[19:16] Source PSR
0 = CPSR 1 = SPSR_<current mode>

[31:28] Condition Field

MRS (transfer register contents to PSR)

31 28	27 23	22	21 12	11 4	3 0
Cond	00010	Pd	101001111	00000000	Rm

[3:0] Source Register

[22] Destination PSR
0 = CPSR 1 = SPSR_<current mode>

[31:28] Condition Field

MRS (transfer register contents or immediate value to PSR flag bits only)

31 28	27 26	25	24 23	22	21 12	11 0
Cond	00	I	10	Pd	101001111	Soucer operand

[22] Destination PSR
0 = CPSR 1 = SPSR_<current mode>

[25] Immediate Operand
0 = Source operand is a register
1 = SPSR_<current mode>

[11:0] Source Operand

11 4	3 0
00000000	Rm

[3:0] Source Register
[11:4] Source operand is an immediate value

11 8	7 0
Rotate	Imm

[7:0] Unsigned 8 bit immediate value
[11:8] Shift applied to Imm

[31:28] Condition Field

Figure 3-11. PSR Transfer

RESERVED BITS

Only twelve bits of the PSR are defined in ARM7TDMI (N,Z,C,V,I,F, T & M[4:0]); the remaining bits are reserved for use in future versions of the processor. Refer to Figure 2-6 for a full description of the PSR bits.

To ensure the maximum compatibility between ARM7TDMI programs and future processors, the following rules should be observed:

- The reserved bits should be preserved when changing the value in a PSR.
- Programs should not rely on specific values from the reserved bits when checking the PSR status, since they may read as one or zero in future processors.

A read-modify-write strategy should therefore be used when altering the control bits of any PSR register; this involves transferring the appropriate PSR register to a general register using the MRS instruction, changing only the relevant bits and then transferring the modified value back to the PSR register using the MSR instruction.

EXAMPLES

The following sequence performs a mode change:

```
MRS     R0,CPSR          ;  Take a copy of the CPSR.
BIC     R0,R0,#0x1F      ;  Clear the mode bits.
ORR     R0,R0,#new_mode  ;  Select new mode
MSR     CPSR,R0          ;  Write back the modified CPSR.
```

When the aim is simply to change the condition code flags in a PSR, a value can be written directly to the flag bits without disturbing the control bits. The following instruction sets the N,Z,C and V flags:

```
MSR     CPSR_flg,#0xF0000000     ;  Set all the flags regardless of their previous state
                                 ;  (does not affect any control bits).
```

No attempt should be made to write an 8 bit immediate value into the whole PSR since such an operation cannot preserve the reserved bits.

INSTRUCTION CYCLE TIMES

PSR transfers take 1S incremental cycles, where S is defined as Sequential (S-cycle).

ASSEMBLY SYNTAX

- MRS - transfer PSR contents to a register
  MRS{cond} Rd,<psr>
- MSR - transfer register contents to PSR
  MSR{cond} <psr>,Rm
- MSR - transfer register contents to PSR flag bits only
  MSR{cond} <psrf>,Rm

The most significant four bits of the register contents are written to the N,Z,C & V flags respectively.

- MSR - transfer immediate value to PSR flag bits only
  MSR{cond} <psrf>,<#expression>

The expression should symbolise a 32 bit value of which the most significant four bits are written to the N,Z,C and V flags respectively.

Key:

{cond}	Two-character condition mnemonic. See Table 3-2..
Rd and Rm	Expressions evaluating to a register number other than R15
<psr>	CPSR, CPSR_all, SPSR or SPSR_all. (CPSR and CPSR_all are synonyms as are SPSR and SPSR_all)
<psrf>	CPSR_flg or SPSR_flg
<#expression>	Where this is used, the assembler will attempt to generate a shifted immediate 8-bit field to match the expression. If this is impossible, it will give an error.

EXAMPLES

In User mode the instructions behave as follows:

```
MSR     CPSR_all,Rm             ;  CPSR[31:28] <- Rm[31:28]
MSR     CPSR_flg,Rm             ;  CPSR[31:28] <- Rm[31:28]
MSR     CPSR_flg,#0xA0000000    ;  CPSR[31:28] <- 0xA (set N,C; clear Z,V)
MRS     Rd,CPSR                 ;  Rd[31:0] <- CPSR[31:0]
```

In privileged modes the instructions behave as follows:

```
MSR     CPSR_all,Rm             ;  CPSR[31:0]  <- Rm[31:0]
MSR     CPSR_flg,Rm             ;  CPSR[31:28] <- Rm[31:28]
MSR     CPSR_flg,#0x50000000    ;  CPSR[31:28] <- 0x5 (set Z,V; clear N,C)
MSR     SPSR_all,Rm             ;  SPSR_<mode>[31:0]<- Rm[31:0]
MSR     SPSR_flg,Rm             ;  SPSR_<mode>[31:28] <- Rm[31:28]
MSR     SPSR_flg,#0xC0000000    ;  SPSR_<mode>[31:28] <- 0xC (set N,Z; clear C,V)
MRS     Rd,SPSR                 ;  Rd[31:0] <- SPSR_<mode>[31:0]
```

MULTIPLY AND MULTIPLY-ACCUMULATE (MUL, MLA)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-12.

The multiply and multiply-accumulate instructions use an 8 bit Booth's algorithm to perform integer multiplication.

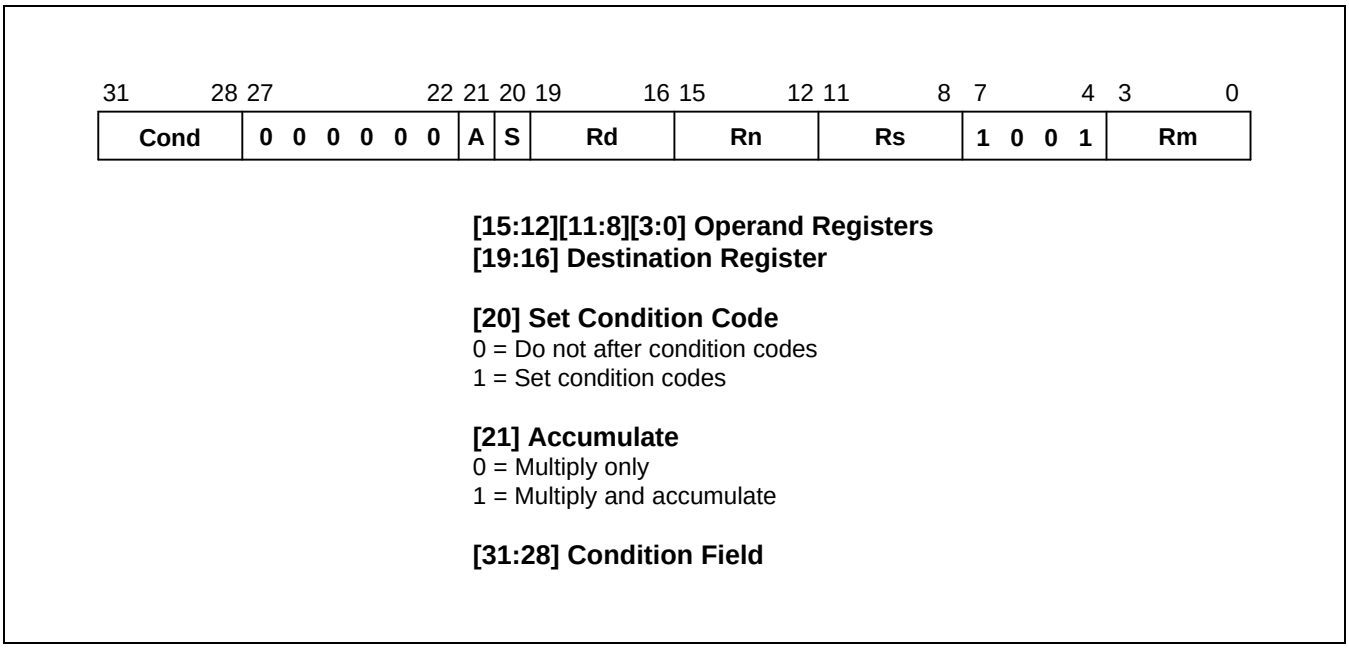


Figure 3-12. Multiply Instructions

The multiply form of the instruction gives Rd:=Rm*Rs. Rn is ignored, and should be set to zero for compatibility with possible future upgrades to the instruction set.The multiply-accumulate form gives Rd:=Rm*Rs+Rn, which can save an explicit ADD instruction in some circumstances. Both forms of the instruction work on operands which may be considered as signed (2's complement) or unsigned integers.

The results of a signed multiply and of an unsigned multiply of 32 bit operands differ only in the upper 32 bits - the low 32 bits of the signed and unsigned results are identical. As these instructions only produce the low 32 bits of a multiply, they can be used for both signed and unsigned multiplies.

For example consider the multiplication of the operands:

Operand A	Operand B	Result
0xFFFFFFF6	0x0000001	0xFFFFFF38

If the Operands Are Interpreted as Signed

Operand A has the value -10, operand B has the value 20, and the result is -200 which is correctly represented as 0xFFFFFF38.

If the Operands Are Interpreted as Unsigned

Operand A has the value 4294967286, operand B has the value 20 and the result is 85899345720, which is represented as 0x13FFFFFF38, so the least significant 32 bits are 0xFFFFFF38.

Operand Restrictions

The destination register Rd must not be the same as the operand register Rm. R15 must not be used as an operand or as the destination register.

All other register combinations will give correct results, and Rd, Rn and Rs may use the same register when required.

CPSR FLAGS

Setting the CPSR flags is optional, and is controlled by the S bit in the instruction. The N (Negative) and Z (Zero) flags are set correctly on the result (N is made equal to bit 31 of the result, and Z is set if and only if the result is zero). The C (Carry) flag is set to a meaningless value and the V (oVerflow) flag is unaffected.

INSTRUCTION CYCLE TIMES

MUL takes 1S + mI and MLA 1S + (m+1)I cycles to execute, where S and I are defined as sequential (S-cycle) and internal (I-cycle), respectively.

m	The number of 8 bit multiplier array cycles is required to complete the multiply, which is controlled by the value of the multiplier operand specified by Rs. Its possible values are as follows
1	If bits [32:8] of the multiplier operand are all zero or all one.
2	If bits [32:16] of the multiplier operand are all zero or all one.
3	If bits [32:24] of the multiplier operand are all zero or all one.
4	In all other cases.

ASSEMBLER SYNTAX

MUL{cond}{S} Rd,Rm,Rs
MLA{cond}{S} Rd,Rm,Rs,Rn

{cond}	Two-character condition mnemonic. See Table 3-2..
{S}	Set condition codes if S present
Rd, Rm, Rs and Rn	Expressions evaluating to a register number other than R15.

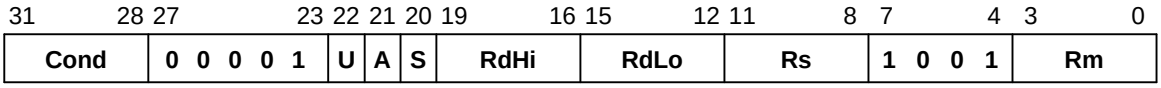
EXAMPLES

```
MUL     R1,R2,R3        ;  R1:=R2*R3
MLAEQS  R1,R2,R3,R4     ;  Conditionally R1:=R2*R3+R4, Setting condition codes.
```

MULTIPLY LONG AND MULTIPLY-ACCUMULATE LONG (MULL, MLAL)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-13.

The multiply long instructions perform integer multiplication on two 32 bit operands and produce 64 bit results. Signed and unsigned multiplication each with optional accumulate give rise to four variations.



[11:8][3:0] Operand Registers
[19:16][15:12] Source Destination Registers

[20] Set Condition Code
0 = Do not after condition codes
1 = Set condition codes

[21] Accumulate
0 = Multiply only
1 = Multiply and accumulate

[22] Unsigned
0 = Unsigned
1 = Signed

[31:28] Condition Field

Figure 3-13. Multiply Long Instructions

The multiply forms (UMULL and SMULL) take two 32 bit numbers and multiply them to produce a 64 bit result of the form RdHi,RdLo := Rm * Rs. The lower 32 bits of the 64 bit result are written to RdLo, the upper 32 bits of the result are written to RdHi.

The multiply-accumulate forms (UMLAL and SMLAL) take two 32 bit numbers, multiply them and add a 64 bit number to produce a 64 bit result of the form RdHi,RdLo := Rm * Rs + RdHi,RdLo. The lower 32 bits of the 64 bit number to add is read from RdLo. The upper 32 bits of the 64 bit number to add is read from RdHi. The lower 32 bits of the 64 bit result are written to RdLo. The upper 32 bits of the 64 bit result are written to RdHi.

The UMULL and UMLAL instructions treat all of their operands as unsigned binary numbers and write an unsigned 64 bit result. The SMULL and SMLAL instructions treat all of their operands as two's-complement signed numbers and write a two's-complement signed 64 bit result.

OPERAND RESTRICTIONS

- R15 must not be used as an operand or as a destination register.
- RdHi, RdLo, and Rm must all specify different registers.

CPSR FLAGS

Setting the CPSR flags is optional, and is controlled by the S bit in the instruction. The N and Z flags are set correctly on the result (N is equal to bit 63 of the result, Z is set if and only if all 64 bits of the result are zero). Both the C and V flags are set to meaningless values.

INSTRUCTION CYCLE TIMES

MULL takes 1S + (m+1)I and MLAL 1S + (m+2)I cycles to execute, where *m* is the number of 8 bit multiplier array cycles required to complete the multiply, which is controlled by the value of the multiplier operand specified by Rs.

Its possible values are as follows:

For Signed INSTRUCTIONS SMULL, SMLAL:

- If bits [31:8] of the multiplier operand are all zero or all one.
- If bits [31:16] of the multiplier operand are all zero or all one.
- If bits [31:24] of the multiplier operand are all zero or all one.
- In all other cases.

For Unsigned Instructions UMULL, UMLAL:

- If bits [31:8] of the multiplier operand are all zero.
- If bits [31:16] of the multiplier operand are all zero.
- If bits [31:24] of the multiplier operand are all zero.
- In all other cases.

S and I are defined as sequential (S-cycle) and internal (I-cycle), respectively.

ASSEMBLER SYNTAX

Table 3-5. Assembler Syntax Descriptions

Mnemonic	Description	Purpose
UMULL{cond}{S} RdLo,RdHi,Rm,Rs	Unsigned Multiply Long	32 x 32 = 64
UMLAL{cond}{S} RdLo,RdHi,Rm,Rs	Unsigned Multiply & Accumulate Long	32 x 32 + 64 = 64
SMULL{cond}{S} RdLo,RdHi,Rm,Rs	Signed Multiply Long	32 x 32 = 64
SMLAL{cond}{S} RdLo,RdHi,Rm,Rs	Signed Multiply & Accumulate Long	32 x 32 + 64 = 64

where:

{cond} Two-character condition mnemonic. See Table 3-2.

{S} Set condition codes if S present

RdLo, RdHi, Rm, Rs Expressions evaluating to a register number other than R15.

EXAMPLES

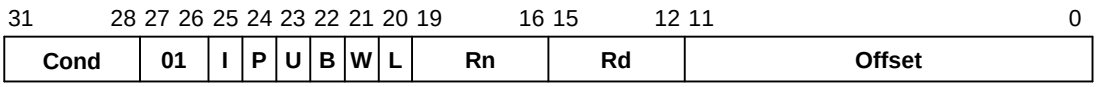
```
UMULL    R1,R4,R2,R3      ; R4,R1:=R2*R3
UMLALS   R1,R5,R2,R3      ; R5,R1:=R2*R3+R5,R1 also setting condition codes
```

SINGLE DATA TRANSFER (LDR, STR)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-14.

The single data transfer instructions are used to load or store single bytes or words of data. The memory address used in the transfer is calculated by adding an offset to or subtracting an offset from a base register.

The result of this calculation may be written back into the base register if auto-indexing is required.



[15:12] Source/Destination Registers

[19:16] Base Register

[20] Load/Store Bit
0 = Store to memory
1 = Load from memory

[21] Write-back Bit
0 = No write-back
1 = Write address into base

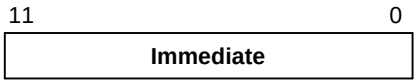
[22] Byte/Word Bit
0 = Transfer word quantity
1 = Transfer byte quantity

[23] Up/Down Bit
0 = Down: subtract offset from base
1 = Up: add offset to base

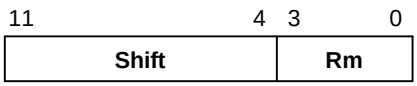
[24] Pre/Post Indexing Bit
0 = Post: add offset after transfer
1 = Pre: add offset bofore transfer

[25] Immediate Offset
0 = Offset is an immediate value

[11:0] Offset



[11:0] Unsigned 12 bit immediate offset



[3:0] Offset register [11:4] Shift applied to Rm

[31:28] Condition Field

Figure 3-14. Single Data Transfer Instructions

OFFSETS AND AUTO-INDEXING

The offset from the base may be either a 12 bit unsigned binary immediate value in the instruction, or a second register (possibly shifted in some way). The offset may be added to (U=1) or subtracted from (U=0) the base register Rn. The offset modification may be performed either before (pre-indexed, P=1) or after (post-indexed, P=0) the base is used as the transfer address.

The W bit gives optional auto increment and decrement addressing modes. The modified base value may be written back into the base (W=1), or the old base value may be kept (W=0). In the case of post-indexed addressing, the write back bit is redundant and is always set to zero, since the old base value can be retained by setting the offset to zero. Therefore post-indexed data transfers always write back the modified base. The only use of the W bit in a post-indexed data transfer is in privileged mode code, where setting the W bit forces non-privileged mode for the transfer, allowing the operating system to generate a user address in a system where the memory management hardware makes suitable use of this hardware.

SHIFTED REGISTER OFFSET

The 8 shift control bits are described in the data processing instructions section. However, the register specified shift amounts are not available in this instruction class. See Figure 3-5.

BYTES AND WORDS

This instruction class may be used to transfer a byte (B=1) or a word (B=0) between an ARM7TDMI register and memory.

The action of LDR(B) and STR(B) instructions is influenced by the **BIGEND** control signal of ARM7TDMI core. The two possible configurations are described below.

NOTE

The KS17C4000 is configured to the big-endian format.

Little-Endian Configuration

A byte load (LDRB) expects the data on data bus inputs 7 through 0 if the supplied address is on a word boundary, on data bus inputs 15 through 8 if it is a word address plus one byte, and so on. The selected byte is placed in the bottom 8 bits of the destination register, and the remaining bits of the register are filled with zeros. Please see Figure 2-2.

A byte store (STRB) repeats the bottom 8 bits of the source register four times across data bus outputs 31 through 0. The external memory system should activate the appropriate byte subsystem to store the data.

A word load (LDR) will normally use a word aligned address. However, an address offset from a word boundary will cause the data to be rotated into the register so that the addressed byte occupies bits 0 to 7. This means that half-words accessed at offsets 0 and 2 from the word boundary will be correctly loaded into bits 0 through 15 of the register. Two shift operations are then required to clear or to sign extend the upper 16 bits.

A word store (STR) should generate a word aligned address. The word presented to the data bus is not affected if the address is not word aligned. That is, bit 31 of the register being stored always appears on data bus output 31.

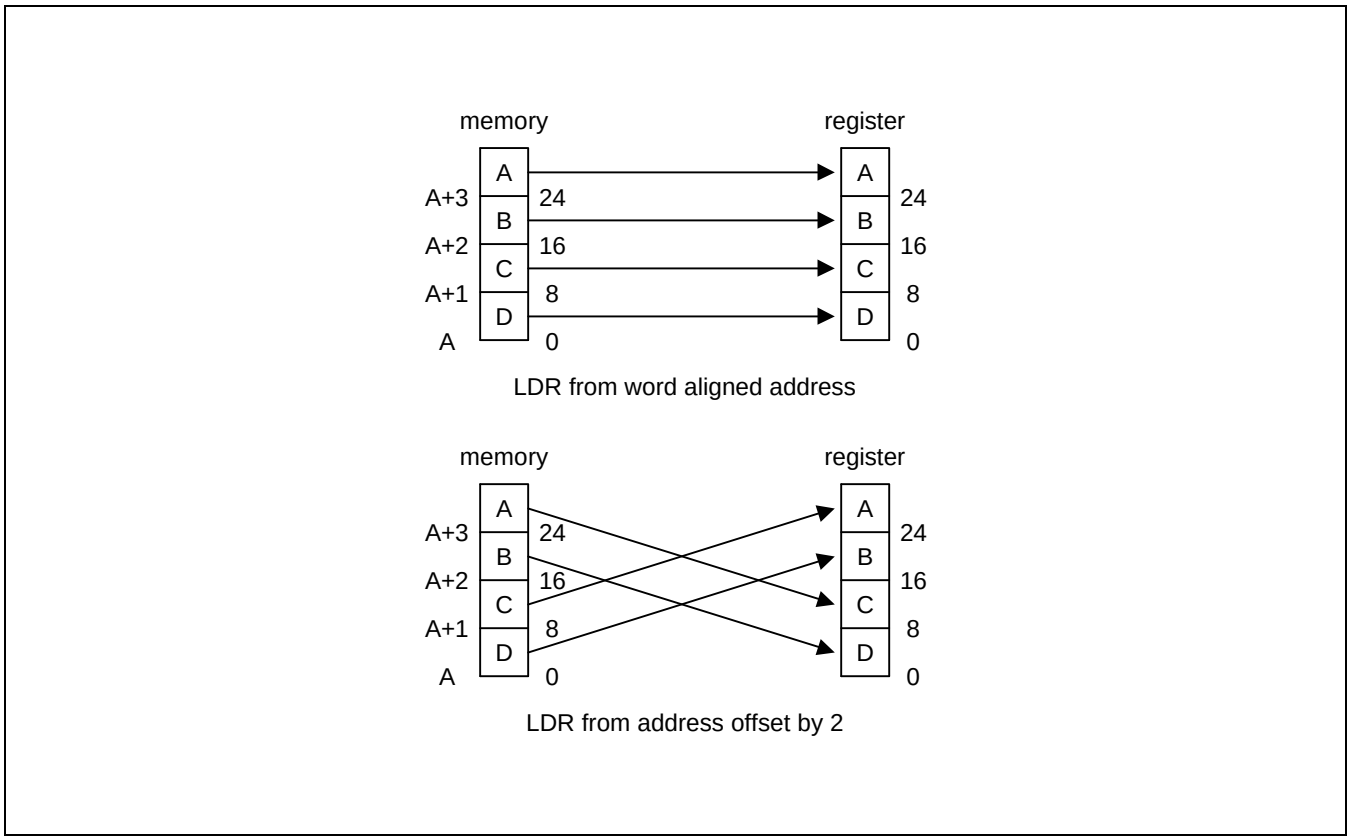


Figure 3-15. Little-Endian Offset Addressing

Big-Endian Configuration

A byte load (LDRB) expects the data on data bus inputs 31 through 24 if the supplied address is on a word boundary, on data bus inputs 23 through 16 if it is a word address plus one byte, and so on. The selected byte is placed in the bottom 8 bits of the destination register and the remaining bits of the register are filled with zeros. Please see Figure 2-1.

A byte store (STRB) repeats the bottom 8 bits of the source register four times across data bus outputs 31 through 0. The external memory system should activate the appropriate byte subsystem to store the data.

A word load (LDR) should generate a word aligned address. An address offset of 0 or 2 from a word boundary will cause the data to be rotated into the register so that the addressed byte occupies bits 31 through 24. This means that half-words accessed at these offsets will be correctly loaded into bits 16 through 31 of the register. A shift operation is then required to move (and optionally sign extend) the data into the bottom 16 bits. An address offset of 1 or 3 from a word boundary will cause the data to be rotated into the register so that the addressed byte occupies bits 15 through 8.

A word store (STR) should generate a word aligned address. The word presented to the data bus is not affected if the address is not word aligned. That is, bit 31 of the register being stored always appears on data bus output 31.

USE OF R15

Write-back must not be specified if R15 is specified as the base register (Rn). When using R15 as the base register you must remember it contains an address 8 bytes on from the address of the current instruction.

R15 must not be specified as the register offset (Rm).

When R15 is the source register (Rd) of a register store (STR) instruction, the stored value will be address of the instruction plus 12.

RESTRICTION ON THE USE OF BASE REGISTER

When configured for late aborts, the following example code is difficult to unwind as the base register, Rn, gets updated before the abort handler starts. Sometimes it may be impossible to calculate the initial value.

After an abort, the following example code is difficult to unwind as the base register, Rn, gets updated before the abort handler starts. Sometimes it may be impossible to calculate the initial value.

Example:

```
LDR     R0,[R1],R1
```

Therefore a post-indexed LDR or STR where Rm is the same register as Rn should not be used.

DATA ABORTS

A transfer to or from a legal address may cause problems for a memory management system. For instance, in a system which uses virtual memory the required data may be absent from main memory. The memory manager can signal a problem by taking the processor ABORT input HIGH whereupon the Data Abort trap will be taken. It is up to the system software to resolve the cause of the problem, then the instruction can be restarted and the original program continued.

INSTRUCTION CYCLE TIMES

Normal LDR instructions take 1S + 1N + 1I and LDR PC take 2S + 2N +1I incremental cycles, where S,N and I are defined as squential (S-cycle), non-sequential (N-cycle), and internal (I-cycle), respectively. STR instructions take 2N incremental cycles to execute.

ASSEMBLER SYNTAX

<LDR|STR>{cond}{B}{T} Rd,<Address>

where:

LDR	Load from memory into a register
STR	Store from a register into memory
{cond}	Two-character condition mnemonic. See Table 3-2.
{B}	If B is present then byte transfer, otherwise word transfer
{T}	If T is present the W bit will be set in a post-indexed instruction, forcing non-privileged mode for the transfer cycle. T is not allowed when a pre-indexed addressing mode is specified or implied.
Rd	An expression evaluating to a valid register number.
Rn and Rm	Expressions evaluating to a register number. If Rn is R15 then the assembler will subtract 8 from the offset value to allow for ARM7TDMI pipelining. In this case base write-back should not be specified.

<Address>can be:

1	An expression which generates an address: The assembler will attempt to generate an instruction using the PC as a base and a corrected immediate offset to address the location given by evaluating the expression. This will be a PC relative, pre-indexed address. If the address is out of range, an error will be generated.	
2	A pre-indexed addressing specification:	
	[Rn]	offset of zero
	[Rn,<#expression>]{!}	offset of <expression> bytes
	[Rn,{+/-}Rm{,<shift>}]{!}	offset of +/- contents of index register, shifted by <shift>
3	A post-indexed addressing specification:	
	[Rn],<#expression>	offset of <expression> bytes
	[Rn],{+/-}Rm{,<shift>}	offset of +/- contents of index register, shifted as by <shift>.
<shift>	General shift operation (see data processing instructions) but you cannot specify the shift amount by a register.	
{!}	Writes back the base register (set the W bit) if! is present.	

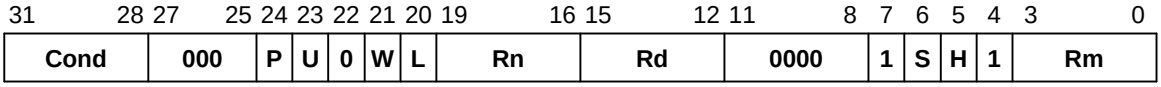
EXAMPLES

```
STR     R1,[R2,R4]!     ; Store R1 at R2+R4 (both of which are registers)
                        ; and write back address to R2.
STR     R1,[R2],R4      ; Store R1 at R2 and write back R2+R4 to R2.
LDR     R1,[R2,#16]     ; Load R1 from contents of R2+16, but don't write back.
LDR     R1,[R2,R3,LSL#2] ; Load R1 from contents of R2+R3*4.
LDREQB  R1,[R6,#5]      ; Conditionally load byte at R6+5 into
                        ; R1 bits 0 to 7, filling bits 8 to 31 with zeros.
STR     R1,PLACE        ; Generate PC relative offset to address PLACE.
PLACE
```

HALFWORD AND SIGNED DATA TRANSFER (LDRH/STRH/LDRSB/LDRSH)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-16.

These instructions are used to load or store half-words of data and also load sign-extended bytes or half-words of data. The memory address used in the transfer is calculated by adding an offset to or subtracting an offset from a base register. The result of this calculation may be written back into the base register if auto-indexing is required.



[3:0] Offset Register

[6][5] S H
0 0 = SWP instruction
0 1 = Unsigned halfwords
1 1 = Signed byte
1 1 = Signed half words

[15:12] Source/Destination Register

[19:16] Base Register

[20] Load/Store
0 = Store to memory
1 = Load from memory

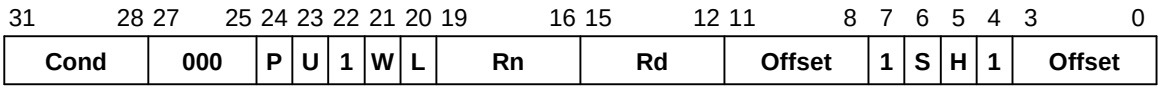
[21] Write-back
0 = No write-back
1 = Write address into base

[23] Up/Down
0 = Down: subtract offset from base
1 = Up: add offset to base

[24] Pre/Post Indexing
0 = Post: add/subtract offset after transfer
1 = Pre: add/subtract offset bofore transfer

[31:28] Condition Field

Figure 3-16. Halfword and Signed Data Transfer with Register Offset



[3:0] Immediate Offset (Low Nibble)

[6][5] S H
0 0 = SWP instruction
0 1 = Unsigned halfwords
1 1 = Signed byte
1 1 = Signed half words

[11:8] Immediate Offset (High Nibble)

[15:12] Source/Destination Register

[19:16] Base Register

[20] Load/Store
0 = Store to memory
1 = Load from memory

[21] Write-back
0 = No write-back
1 = Write address into base

[23] Up/Down
0 = Down: subtract offset from base
1 = Up: add offset to base

[24] Pre/Post Indexing
0 = Post: add/subtract offset after transfer
1 = Pre: add/subtract offset bofore transfer

[31:28] Condition Field

Figure 3-17. Halfword and Signed Data Transfer with Immediate Offset and Auto-Indexing

OFFSETS AND AUTO-INDEXING

The offset from the base may be either a 8-bit unsigned binary immediate value in the instruction, or a second register. The 8-bit offset is formed by concatenating bits 11 to 8 and bits 3 to 0 of the instruction word, such that bit 11 becomes the MSB and bit 0 becomes the LSB. The offset may be added to (U=1) or subtracted from (U=0) the base register Rn. The offset modification may be performed either before (pre-indexed, P=1) or after (post-indexed, P=0) the base register is used as the transfer address.

The W bit gives optional auto-increment and decrement addressing modes. The modified base value may be written back into the base (W=1), or the old base may be kept (W=0). In the case of post-indexed addressing, the write back bit is redundant and is always set to zero, since the old base value can be retained if necessary by setting the offset to zero. Therefore post-indexed data transfers always write back the modified base.

The Write-back bit should not be set high (W=1) when post-indexed addressing is selected.

HALFWORD LOAD AND STORES

Setting S=0 and H=1 may be used to transfer unsigned Half-words between an ARM7TDMI register and memory.

The action of LDRH and STRH instructions is influenced by the BIGEND control signal. The two possible configurations are described in the section below.

SIGNED BYTE AND HALFWORD LOADS

The S bit controls the loading of sign-extended data. When S=1 the H bit selects between Bytes (H=0) and Half-words (H=1). The L bit should not be set low (Store) when Signed (S=1) operations have been selected.

The LDRSB instruction loads the selected Byte into bits 7 to 0 of the destination register and bits 31 to 8 of the destination register are set to the value of bit 7, the sign bit.

The LDRSH instruction loads the selected Half-word into bits 15 to 0 of the destination register and bits 31 to 16 of the destination register are set to the value of bit 15, the sign bit.

The action of the LDRSB and LDRSH instructions is influenced by the BIGEND control signal. The two possible configurations are described in the following section.

ENDIANNESS AND BYTE/HALFWORD SELECTION

Little-Endian Configuration

A signed byte load (LDRSB) expects data on data bus inputs 7 through to 0 if the supplied address is on a word boundary, on data bus inputs 15 through to 8 if it is a word address plus one byte, and so on. The selected byte is placed in the bottom 8 bit of the destination register, and the remaining bits of the register are filled with the sign bit, bit 7 of the byte. Please see Figure 2-2.

A halfword load (LDRSH or LDRH) expects data on data bus inputs 15 through to 0 if the supplied address is on a word boundary and on data bus inputs 31 through to 16 if it is a halfword boundary, (A[1]=1).The supplied address should always be on a halfword boundary. If bit 0 of the supplied address is HIGH then the ARM7TDMI will load an unpredictable value. The selected halfword is placed in the bottom 16 bits of the destination register. For unsigned half-words (LDRH), the top 16 bits of the register are filled with zeros and for signed half-words (LDRSH) the top 16 bits are filled with the sign bit, bit 15 of the halfword.

A halfword store (STRH) repeats the bottom 16 bits of the source register twice across the data bus outputs 31 through to 0. The external memory system should activate the appropriate halfword subsystem to store the data. Note that the address must be halfword aligned, if bit 0 of the address is HIGH this will cause unpredictable behaviour.

Big-Endian Configuration

A signed byte load (LDRSB) expects data on data bus inputs 31 through to 24 if the supplied address is on a word boundary, on data bus inputs 23 through to 16 if it is a word address plus one byte, and so on. The selected byte is placed in the bottom 8 bit of the destination register, and the remaining bits of the register are filled with the sign bit, bit 7 of the byte. Please see Figure 2-1.

A halfword load (LDRSH or LDRH) expects data on data bus inputs 31 through to 16 if the supplied address is on a word boundary and on data bus inputs 15 through to 0 if it is a halfword boundary, (A[1]=1). The supplied address should always be on a halfword boundary. If bit 0 of the supplied address is HIGH then the ARM7TDMI will load an unpredictable value. The selected halfword is placed in the bottom 16 bits of the destination register. For unsigned half-words (LDRH), the top 16 bits of the register are filled with zeros and for signed half-words (LDRSH) the top 16 bits are filled with the sign bit, bit 15 of the halfword.

A halfword store (STRH) repeats the bottom 16 bits of the source register twice across the data bus outputs 31 through to 0. The external memory system should activate the appropriate halfword subsystem to store the data. Note that the address must be halfword aligned, if bit 0 of the address is HIGH this will cause unpredictable behaviour.

NOTE

The KS17C4000 is configured to the big-endian format.

USE OF R15

Write-back should not be specified if R15 is specified as the base register (Rn). When using R15 as the base register you must remember it contains an address 8 bytes on from the address of the current instruction.

R15 should not be specified as the register offset (Rm).

When R15 is the source register (Rd) of a Half-word store (STRH) instruction, the stored address will be address of the instruction plus 12.

DATA ABORTS

A transfer to or from a legal address may cause problems for a memory management system. For instance, in a system which uses virtual memory the required data may be absent from the main memory. The memory manager can signal a problem by taking the processor ABORT input HIGH whereupon the Data Abort trap will be taken. It is up to the system software to resolve the cause of the problem, then the instruction can be restarted and the original program continued.

INSTRUCTION CYCLE TIMES

Normal LDR(H,SH,SB) instructions take 1S + 1N + 1I. LDR(H,SH,SB) PC take 2S + 2N + 1I incremental cycles. S,N and I are defined as squential (S-cycle), non-squential (N-cycle), and internal (I-cycle), respectively. STRH instructions take 2N incremental cycles to execute.

ASSEMBLER SYNTAX

<LDR|STR>{cond}<H|SH|SB> Rd,<address>

LDR	Load from memory into a register
STR	Store from a register into memory
{cond}	Two-character condition mnemonic. See Table 3-2..
H	Transfer halfword quantity
SB	Load sign extended byte (Only valid for LDR)
SH	Load sign extended halfword (Only valid for LDR)
Rd	An expression evaluating to a valid register number.

<address> can be:

1 An expression which generates an address:
The assembler will attempt to generate an instruction using the PC as a base and a corrected immediate offset to address the location given by evaluating the expression. This will be a PC relative, pre-indexed address. If the address is out of range, an error will be generated.

2 A pre-indexed addressing specification:

[Rn]	offset of zero
[Rn,<#expression>]{!}	offset of <expression> bytes
[Rn,{+/-}Rm]{!}	offset of +/- contents of index register

3 A post-indexed addressing specification:

[Rn],<#expression>	offset of <expression> bytes
[Rn],{+/-}Rm	offset of +/- contents of index register.

4 Rn and Rm are expressions evaluating to a register number. If Rn is R15 then the assembler will subtract 8 from the offset value to allow for ARM7TDMI pipelining. In this case base write-back should not be specified.

{!} Writes back the base register (set the W bit) if ! is present.

EXAMPLES

```
LDRH    R1,[R2,-R3]!        ;  Load R1 from the contents of the halfword address
                            ;  contained in R2-R3 (both of which are registers)
                            ;  and write back address to R2
STRH    R3,[R4,#14]         ;  Store the halfword in R3 at R14+14 but don't write back.
LDRSB   R8,[R2],#-223       ;  Load R8 with the sign extended contents of the byte
                            ;  address contained in R2 and write back R2-223 to R2.
LDRNESH R11,[R0]            ;  Conditionally load R11 with the sign extended contents
                            ;  of the halfword address contained in R0.
HERE                        ;  Generate PC relative offset to address FRED.
STRH    R5, [PC,#(FRED-HERE-8)];  Store the halfword in R5 at address FRED
FRED
```

BLOCK DATA TRANSFER (LDM, STM)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-18.

Block data transfer instructions are used to load (LDM) or store (STM) any subset of the currently visible registers. They support all possible stacking modes, maintaining full or empty stacks which can grow up or down memory, and are very efficient instructions for saving or restoring context, or for moving large blocks of data around main memory.

THE REGISTER LIST

The instruction can cause the transfer of any registers in the current bank (and non-user mode programs can also transfer to and from the user bank, see below). The register list is a 16 bit field in the instruction, with each bit corresponding to a register. A 1 in bit 0 of the register field will cause R0 to be transferred, a 0 will cause it not to be transferred; similarly bit 1 controls the transfer of R1, and so on.

Any subset of the registers, or all the registers, may be specified. The only restriction is that the register list should not be empty.

Whenever R15 is stored to memory the stored value is the address of the STM instruction plus 12.

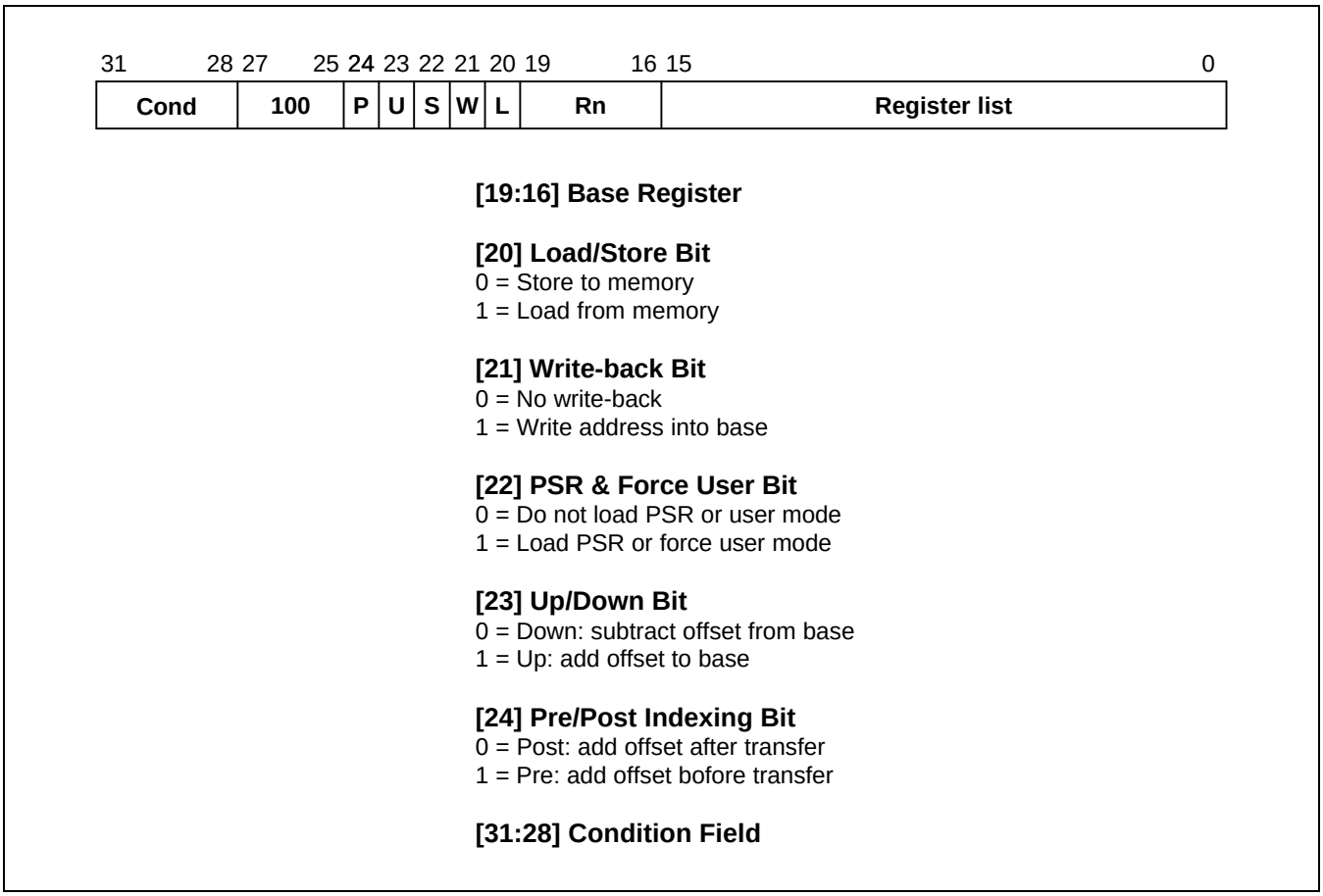


Figure 3-18. Block Data Transfer Instructions

ADDRESSING MODES

The transfer addresses are determined by the contents of the base register (Rn), the pre/post bit (P) and the up/down bit (U). The registers are transferred in the order lowest to highest, so R15 (if in the list) will always be transferred last. The lowest register also gets transferred to/from the lowest memory address. By way of illustration, consider the transfer of R1, R5 and R7 in the case where Rn=0x1000 and write back of the modified base is required (W=1). Figure 3.19-22 show the sequence of register transfers, the addresses used, and the value of Rn after the instruction has completed.

In all cases, had write back of the modified base not been required (W=0), Rn would have retained its initial value of 0x1000 unless it was also in the transfer list of a load multiple register instruction, when it would have been overwritten with the loaded value.

ADDRESS ALIGNMENT

The address should normally be a word aligned quantity and non-word aligned addresses do not affect the instruction. However, the bottom 2 bits of the address will appear on **A[1:0]** and might be interpreted by the memory system.

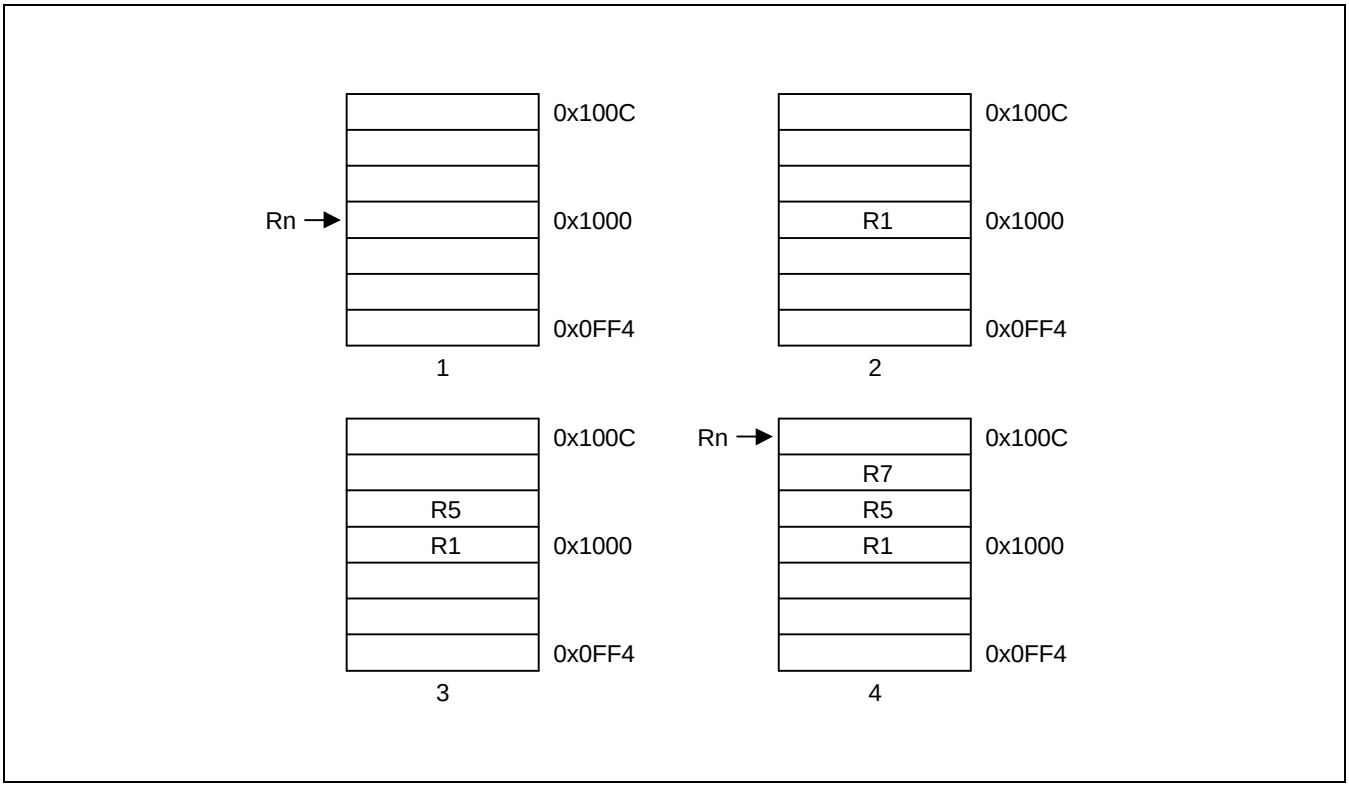


Figure 3-19. Post-Increment Addressing

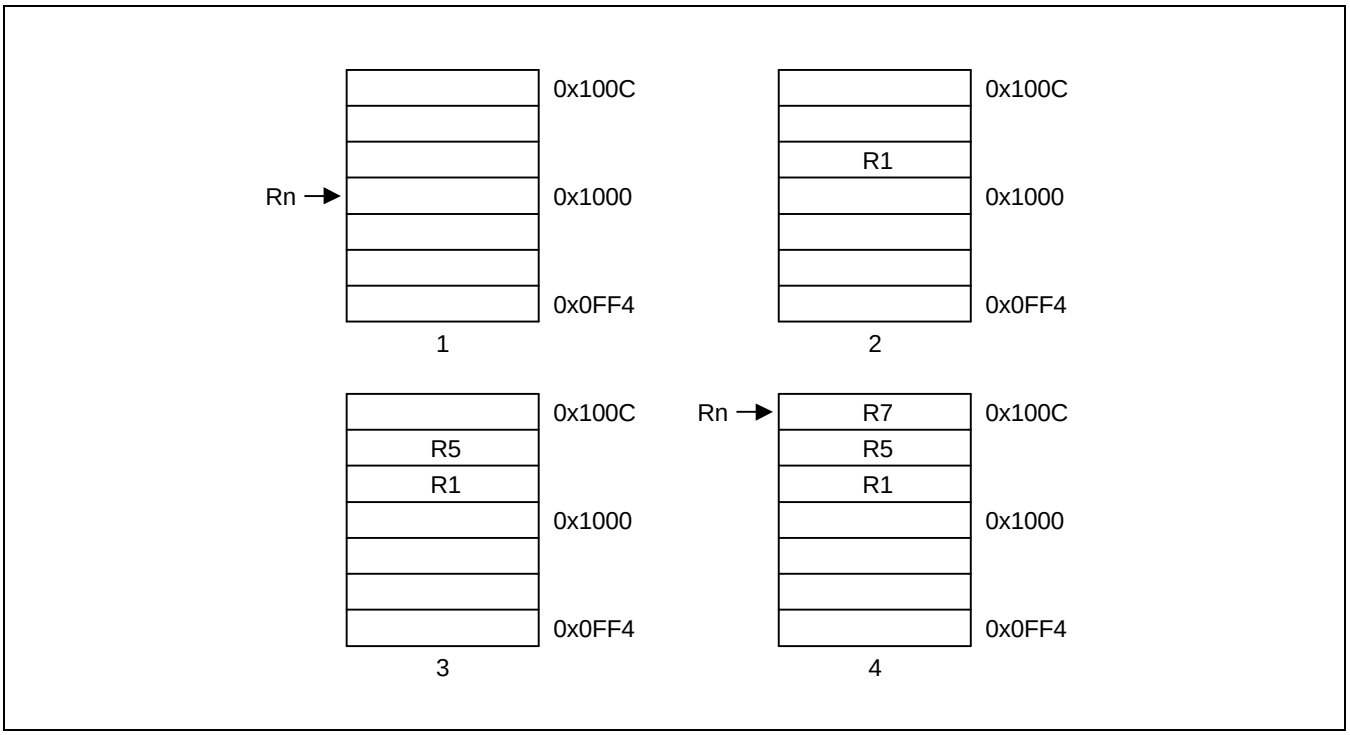


Figure 3-20. Pre-Increment Addressing

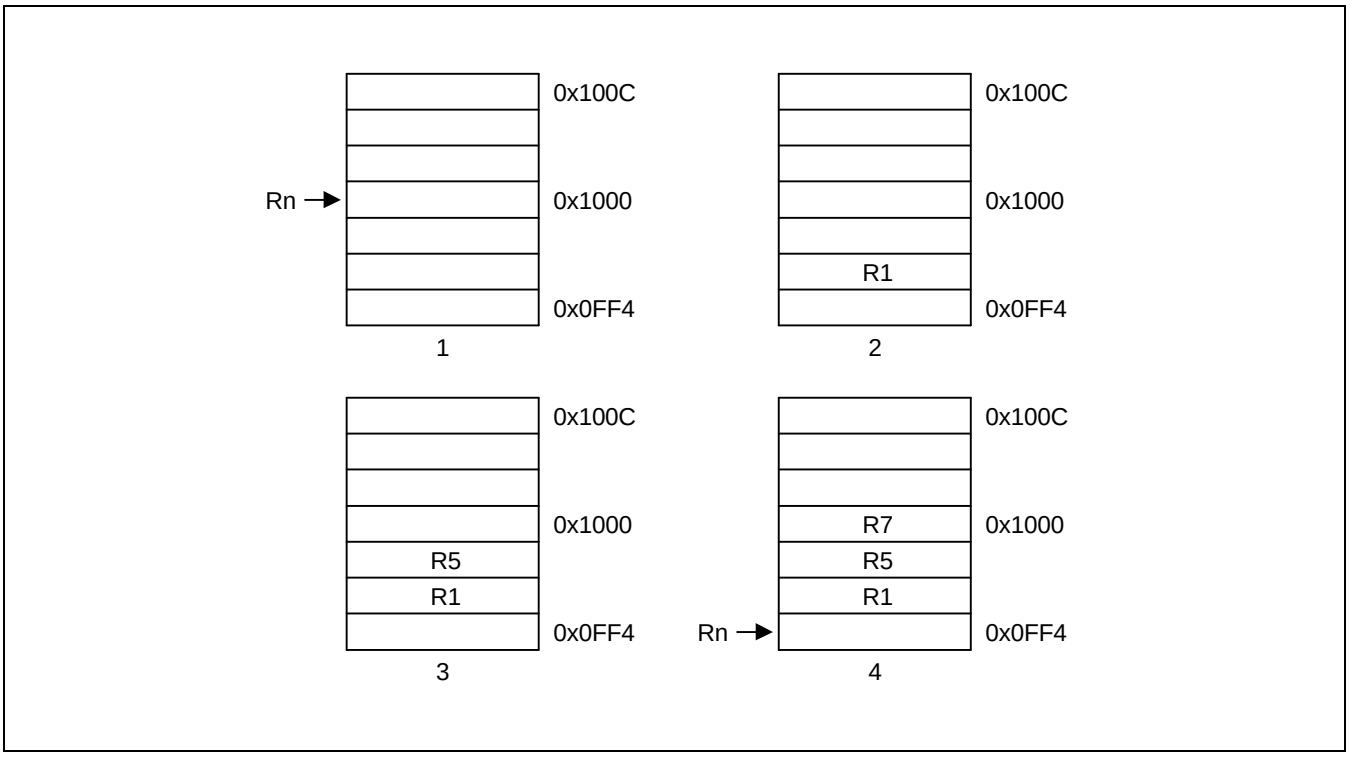


Figure 3-21. Post-Decrement Addressing

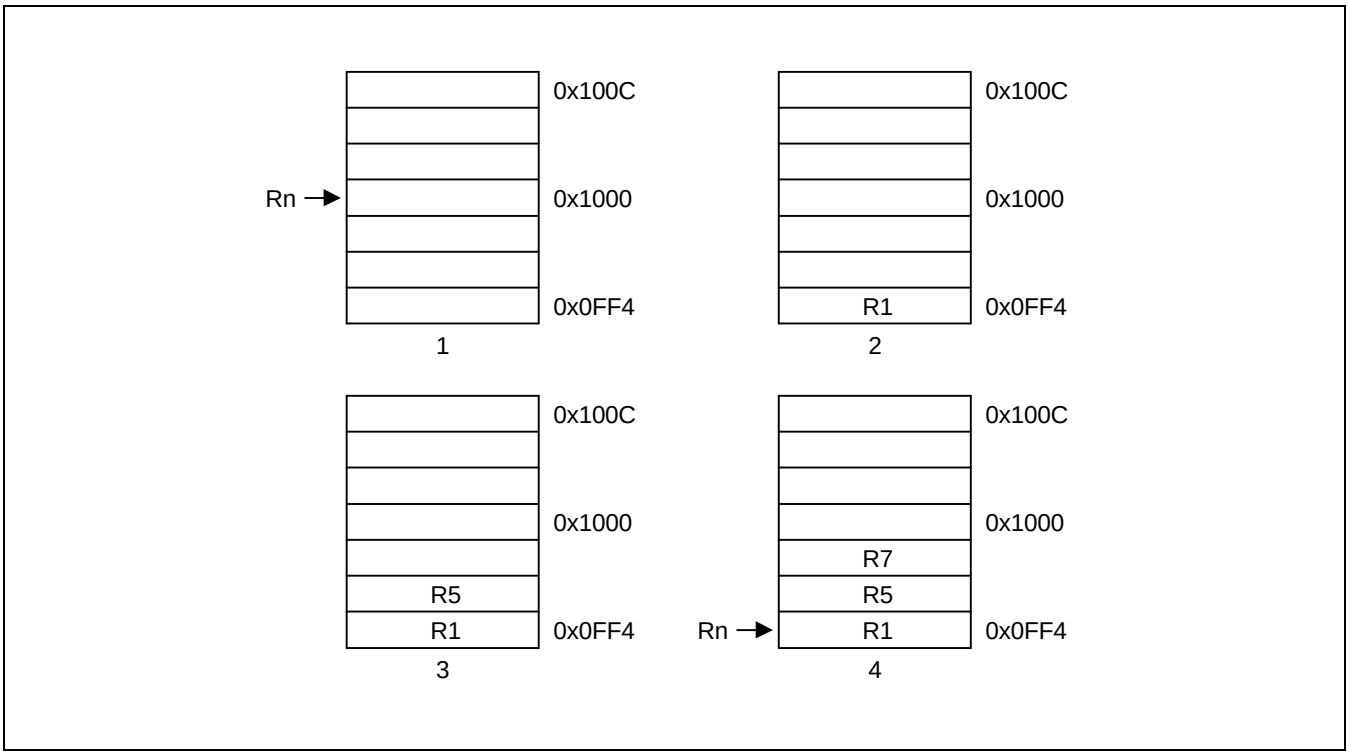


Figure 3-22. Pre-Decrement Addressing

USE OF THE S BIT

When the S bit is set in a LDM/STM instruction its meaning depends on whether or not R15 is in the transfer list and on the type of instruction. The S bit should only be set if the instruction is to execute in a privileged mode.

LDM with R15 in Transfer List and S Bit Set (Mode Changes)

If the instruction is a LDM then SPSR_<mode> is transferred to CPSR at the same time as R15 is loaded.

STM with R15 in Transfer List and S Bit Set (User Bank Transfer)

The registers transferred are taken from the User bank rather than the bank corresponding to the current mode. This is useful for saving the user state on process switches. Base write-back should not be used when this mechanism is employed.

R15 not in List and S Bit Set (User Bank Transfer)

For both LDM and STM instructions, the User bank registers are transferred rather than the register bank corresponding to the current mode. This is useful for saving the user state on process switches. Base write-back should not be used when this mechanism is employed.

When the instruction is LDM, care must be taken not to read from a banked register during the following cycle (inserting a dummy instruction such as MOV R0, R0 after the LDM will ensure safety).

USE OF R15 AS THE BASE

R15 should not be used as the base register in any LDM or STM instruction.

INCLUSION OF THE BASE IN THE REGISTER LIST

When write-back is specified, the base is written back at the end of the second cycle of the instruction. During a STM, the first register is written out at the start of the second cycle. A STM which includes storing the base, with the base as the first register to be stored, will therefore store the unchanged value, whereas with the base second or later in the transfer order, will store the modified value. A LDM will always overwrite the updated base if the base is in the list.

DATA ABORTS

Some legal addresses may be unacceptable to a memory management system, and the memory manager can indicate a problem with an address by taking the **ABORT** signal HIGH. This can happen on any transfer during a multiple register load or store, and must be recoverable if ARM7TDMI is to be used in a virtual memory system.

Abort during STM Instructions

If the abort occurs during a store multiple instruction, ARM7TDMI takes little action until the instruction completes, whereupon it enters the data abort trap. The memory manager is responsible for preventing erroneous writes to the memory. The only change to the internal state of the processor will be the modification of the base register if write-back was specified, and this must be reversed by software (and the cause of the abort resolved) before the instruction may be retried.

Aborts during LDM Instructions

When ARM7TDMI detects a data abort during a load multiple instruction, it modifies the operation of the instruction to ensure that recovery is possible.

- Overwriting of registers stops when the abort happens. The aborting load will not take place but earlier ones may have overwritten registers. The PC is always the last register to be written and so will always be preserved.
- The base register is restored, to its modified value if write-back was requested. This ensures recoverability in the case where the base register is also in the transfer list, and may have been overwritten before the abort occurred.

The data abort trap is taken when the load multiple has completed, and the system software must undo any base modification (and resolve the cause of the abort) before restarting the instruction.

INSTRUCTION CYCLE TIMES

Normal LDM instructions take nS + 1N + 1I and LDM PC takes (n+1)S + 2N + 1I incremental cycles, where S,N and I are defined as squential (S-cycle), non-sequential (N-cycle), and internal (I-cycle), respectively. STM instructions take (n-1)S + 2N incremental cycles to execute, where *n* is the number of words transferred.

ASSEMBLER SYNTAX

<LDM|STM>{cond}<FD|ED|FA|EA|IA|IB|DA|DB> Rn{!},<Rlist>{^}

where:

{cond}	Two character condition mnemonic. See Table 3-2.
Rn	An expression evaluating to a valid register number
<Rlist>	A list of registers and register ranges enclosed in {} (e.g. {R0,R2-R7,R10}).
{!}	If present requests write-back (W=1), otherwise W=0.
{^}	If present set S bit to load the CPSR along with the PC, or force transfer of user bank when in privileged mode.

Addressing Mode Names

There are different assembler mnemonics for each of the addressing modes, depending on whether the instruction is being used to support stacks or for other purposes. The equivalence between the names and the values of the bits in the instruction are shown in the following table 3-6.

Table 3-6. Addressing Mode Names

Name	Stack	Other	L bit	P bit	U bit
Pre-Increment Load	LDMED	LDMIB	1	1	1
Post-Increment Load	LDMFD	LDMIA	1	0	1
Pre-Increment Load	LDMEA	LDMDB	1	1	0
Post-Increment Load	LDMFA	LDMDA	1	0	0
Pre-Increment Load	STMFA	STMIB	0	1	1
Post-Increment Load	STMEA	STMIA	0	0	1
Pre-Increment Load	STMFD	STMDB	0	1	0
Post-Increment Load	STMED	STMDA	0	0	0

FD, ED, FA, EA define pre/post indexing and the up/down bit by reference to the form of stack required. The F and E refer to a "full" or "empty" stack, i.e. whether a pre-index has to be done (full) before storing to the stack. The A and D refer to whether the stack is ascending or descending. If ascending, a STM will go up and LDM down, if descending, vice-versa.

IA, IB, DA, DB allow control when LDM/STM are not being used for stacks and simply mean Increment After, Increment Before, Decrement After, Decrement Before.

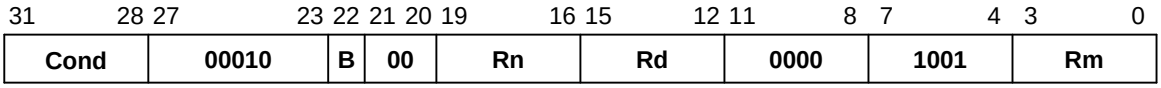
EXAMPLES

```
LDMFD   SP!,{R0,R1,R2}    ;  Unstack 3 registers.
STMIA   R0,{R0-R15}       ;  Save all registers.
LDMFD   SP!,{R15}         ;  R15 ← (SP), CPSR unchanged.
LDMFD   SP!,{R15}^        ;  R15 ← (SP), CPSR <- SPSR_mode
                          ;  (allowed only in privileged modes).
STMFD   R13,{R0-R14}^     ;  Save user mode regs on stack
                          ;  (allowed only in privileged modes).
```

These instructions may be used to save state on subroutine entry, and restore it efficiently on return to the calling routine:

```
STMED   SP!,{R0-R3,R14}   ;  Save R0 to R3 to use as workspace
                          ;  and R14 for returning.
BL      somewhere         ;  This nested call will overwrite R14
LDMED   SP!,{R0-R3,R15}   ;  Restore workspace and return.
```

SINGLE DATA SWAP (SWP)



[3:0] Source Register

[15:12] Destination Register

[19:16] Base Register

[22] Byte/Word Bit
0 = Swap word quantity
1 = Swap word quantity

[31:28] Condition Field

Figure 3-23. Swap Instruction

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-23.

The data swap instruction is used to swap a byte or word quantity between a register and external memory. This instruction is implemented as a memory read followed by a memory write which are “locked” together (the processor cannot be interrupted until both operations have completed, and the memory manager is warned to treat them as inseparable). This class of instruction is particularly useful for implementing software semaphores.

The swap address is determined by the contents of the base register (Rn). The processor first reads the contents of the swap address. Then it writes the contents of the source register (Rm) to the swap address, and stores the old memory contents in the destination register (Rd). The same register may be specified as both the source and destination.

The **LOCK** output goes HIGH for the duration of the read and write operations to signal to the external memory manager that they are locked together, and should be allowed to complete without interruption. This is important in multi-processor systems where the swap instruction is the only indivisible instruction which may be used to implement semaphores; control of the memory must not be removed from a processor while it is performing a locked operation.

BYTES AND WORDS

This instruction class may be used to swap a byte (B=1) or a word (B=0) between an ARM7TDMI register and memory. The SWP instruction is implemented as a LDR followed by a STR and the action of these is as described in the section on single data transfers. In particular, the description of Big and Little Endian configuration applies to the SWP instruction.

USE OF R15

Do not use R15 as an operand (Rd, Rn or Rs) in a SWP instruction.

DATA ABORTS

If the address used for the swap is unacceptable to a memory management system, the memory manager can flag the problem by driving ABORT HIGH. This can happen on either the read or the write cycle (or both), and in either case, the Data Abort trap will be taken. It is up to the system software to resolve the cause of the problem, then the instruction can be restarted and the original program continued.

INSTRUCTION CYCLE TIMES

Swap instructions take 1S + 2N +1I incremental cycles to execute, where S,N and I are defined as squential (S-cycle), non-sequential, and internal (I-cycle), respectively.

ASSEMBLER SYNTAX

<SWP>{cond}{B} Rd,Rm,[Rn]

{cond}	Two-character condition mnemonic. See Table 3-2.
{B}	If B is present then byte transfer, otherwise word transfer
Rd,Rm,Rn	Expressions evaluating to valid register numbers

EXAMPLES

```
SWP     R0,R1,[R2]    ; Load R0 with the word addressed by R2, and
                      ; store R1 at R2.
SWPB    R2,R3,[R4]    ; Load R2 with the byte addressed by R4, and
                      ; store bits 0 to 7 of R3 at R4.
SWPEQ   R0,R0,[R1]    ; Conditionally swap the contents of the
                      ; word addressed by R1 with R0.
```

SOFTWARE INTERRUPT (SWI)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-24, below.

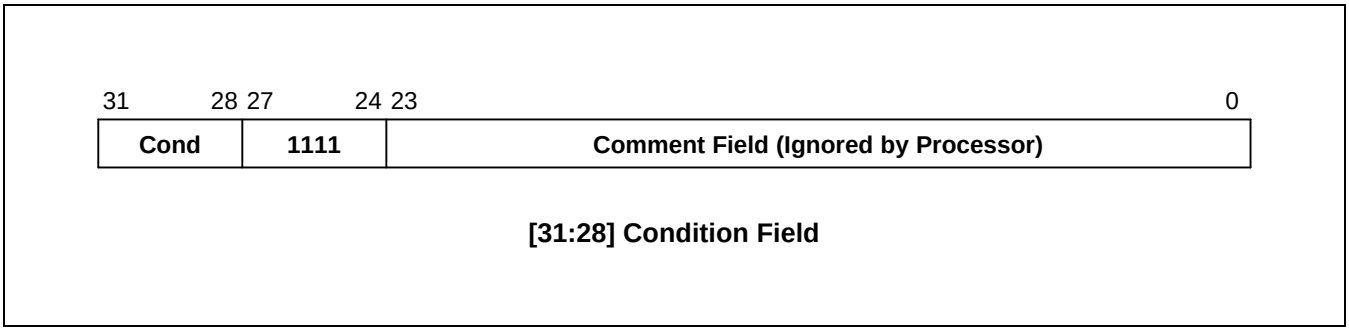


Figure 3-24. Software Interrupt Instruction

The software interrupt instruction is used to enter Supervisor mode in a controlled manner. The instruction causes the software interrupt trap to be taken, which effects the mode change. The PC is then forced to a fixed value (0x08) and the CPSR is saved in SPSR_svc. If the SWI vector address is suitably protected (by external memory management hardware) from modification by the user, a fully protected operating system may be constructed.

RETURN FROM THE SUPERVISOR

The PC is saved in R14_svc upon entering the software interrupt trap, with the PC adjusted to point to the word after the SWI instruction. MOVS PC,R14_svc will return to the calling program and restore the CPSR.

Note that the link mechanism is not re-entrant, so if the supervisor code wishes to use software interrupts within itself it must first save a copy of the return address and SPSR.

COMMENT FIELD

The bottom 24 bits of the instruction are ignored by the processor, and may be used to communicate information to the supervisor code. For instance, the supervisor may look at this field and use it to index into an array of entry points for routines which perform the various supervisor functions.

INSTRUCTION CYCLE TIMES

Software interrupt instructions take 2S + 1N incremental cycles to execute, where S and N are defined as sequential (S-cycle) and non-sequential (N-cycle).

ASSEMBLER SYNTAX

SWI{cond} <expression>

{cond}	Two character condition mnemonic, Table 3-2.
<expression>	Evaluated and placed in the comment field (which is ignored by ARM7TDMI).

EXAMPLES

```
        SWI     ReadC           ;  Get next character from read stream.
        SWI     WriteI+"k"      ;  Output a "k" to the write stream.
        SWINE   0               ;  Conditionally call supervisor with 0 in comment field.
```

Supervisor code

The previous examples assume that suitable supervisor code exists, for instance:

```
        0x08 B Supervisor               ;  SWI entry point
        EntryTable                      ;  Addresses of supervisor routines
        DCD ZeroRtn
        DCD ReadCRtn
        DCD WriteIRtn
        • • •
        Zero        EQU  0
ReadC   EQU  256
WriteI  EQU  512

        Supervisor                      ;  SWI has routine required in bits 8-23 and data (if any) in
                                        ;  bits 0–7. Assumes R13_svc points to a suitable stack
        STMFD   R13,{R0-R2,R14}         ;  Save work registers and return address.
        LDR     R0,[R14,#-4]            ;  Get SWI instruction.
        BIC     R0,R0,#0xFF000000       ;  Clear top 8 bits.
        MOV     R1,R0,LSR#8             ;  Get routine offset.
        ADR     R2,EntryTable           ;  Get start address of entry table.
        LDR     R15,[R2,R1,LSL#2]       ;  Branch to appropriate routine.
        WriteIRtn                       ;  Enter with character in R0 bits 0-7.
        • • •
        LDMFD   R13,{R0-R2,R15}^        ;  Restore workspace and return,
                                        ;  restoring processor mode and flags.
```

COPROCESSOR DATA OPERATIONS (CDP)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-25.

This class of instruction is used to tell a coprocessor to perform some internal operation. No result is communicated back to ARM7TDMI, and it will not wait for the operation to complete. The coprocessor could contain a queue of such instructions awaiting execution, and their execution can overlap other activity, allowing the coprocessor and ARM7TDMI to perform independent tasks in parallel.

COPROCESSOR INSTRUCTIONS

The KS17C4000, unlike some other ARM-based processors, does not have an external coprocessor interface. It does not have a on-chip coprocessor also.

So then all coprocessor instructions will cause the undefinded instruction trap to be taken on the KS17C4000. These coprocessor instructions can be emulated by the undefined trap handler. Even though external coprocessor can not be connected to the KS17C4000, the coprocessor instructions are still described here in full for completeness. (Remember that any external coprocessor described in this section is a software emulation.)

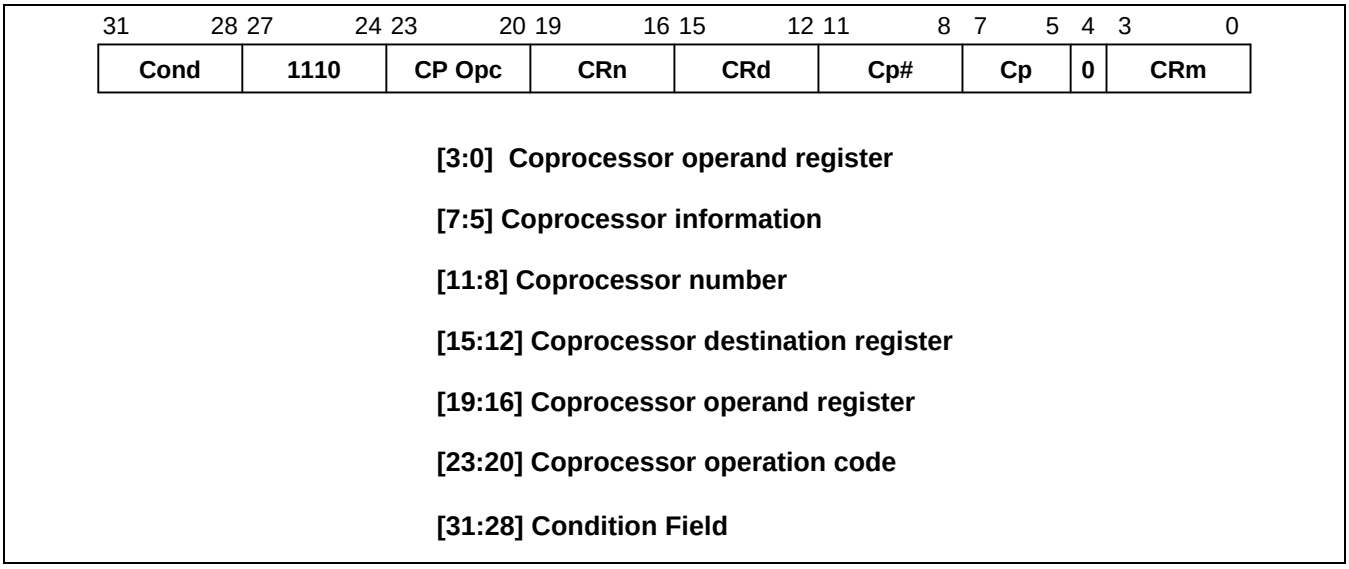


Figure 3-25. Coprocessor Data Operation Instruction

THE COPROCESSOR FIELDS

Only bit 4 and bits 24 to 31 are significant to ARM7TDMI. The remaining bits are used by coprocessors. The above field names are used by convention, and particular coprocessors may redefine the use of all fields except CP# as appropriate. The CP# field is used to contain an identifying number (in the range 0 to 15) for each coprocessor, and a coprocessor will ignore any instruction which does not contain its number in the CP# field.

The conventional interpretation of the instruction is that the coprocessor should perform an operation specified in the CP Opc field (and possibly in the CP field) on the contents of CRn and CRm, and place the result in CRd.

INSTRUCTION CYCLE TIMES

Coprocessor data operations take 1S + bI incremental cycles to execute, where *b* is the number of cycles spent in the coprocessor busy-wait loop.

S and I are defined as squential (S-cycle) and internal (I-cycle).

ASSEMBLER SYNTAX

CDP{cond} p#,<expression1>,cd,cn,cm{,<expression2>}

{cond}	Two character condition mnemonic. See Table 3-2.
p#	The unique number of the required coprocessor
<expression1>	Evaluated to a constant and placed in the CP Opc field
cd, cn and cm	Evaluate to the valid coprocessor register numbers CRd, CRn and CRm respectively
<expression2>	Where present is evaluated to a constant and placed in the CP field

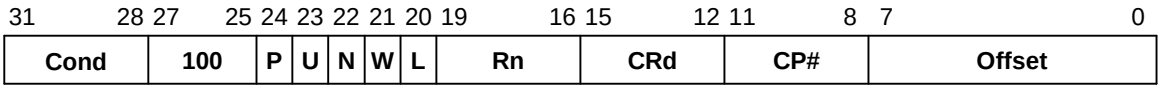
EXAMPLES

```
CDP     p1,10,c1,c2,c3      ; Request coproc 1 to do operation 10
                            ; on CR2 and CR3, and put the result in CR1.
CDPEQ   p2,5,c1,c2,c3,2     ; If Z flag is set request coproc 2 to do operation 5 (type 2)
                            ; on CR2 and CR3, and put the result in CR1.
```

COPROCESSOR DATA TRANSFERS (LDC, STC)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-26.

This class of instruction is used to load (LDC) or store (STC) a subset of a coprocessors's registers directly to memory. ARM7TDMI is responsible for supplying the memory address, and the coprocessor supplies or accepts the data and controls the number of words transferred.



[7:0] Unsigned 8 Bit Immediate Offset

[11:8] Coprocessor Number

[15:12] Coprocessor Source/Destination Register

[19:16] Base Register

[20] Load/Store Bit
0 = Store to memory
1 = Load from memory

[21] Write-back Bit
0 = No write-back
1 = Write address into base

[22] Transfer Length

[23] Up/Down Bit
0 = Down: subtract offset from base
1 = Up: add offset to base

[24] Pre/Post Indexing Bit
0 = Post: add offset after transfer
1 = Pre: add offset bofore transfer

[31:28] Condition Field

Figure 3-26. Coprocessor Data Transfer Instructions

THE COPROCESSOR FIELDS

The CP# field is used to identify the coprocessor which is required to supply or accept the data, and a coprocessor will only respond if its number matches the contents of this field.

The CRd field and the N bit contain information for the coprocessor which may be interpreted in different ways by different coprocessors, but by convention CRd is the register to be transferred (or the first register where more than one is to be transferred), and the N bit is used to choose one of two transfer length options. For instance N=0 could select the transfer of a single register, and N=1 could select the transfer of all the registers for context switching.

ADDRESSING MODES

ARM7TDMI is responsible for providing the address used by the memory system for the transfer, and the addressing modes available are a subset of those used in single data transfer instructions. Note, however, that the immediate offsets are 8 bits wide and specify word offsets for coprocessor data transfers, whereas they are 12 bits wide and specify byte offsets for single data transfers.

The 8 bit unsigned immediate offset is shifted left 2 bits and either added to (U=1) or subtracted from (U=0) the base register (Rn); this calculation may be performed either before (P=1) or after (P=0) the base is used as the transfer address. The modified base value may be overwritten back into the base register (if W=1), or the old value of the base may be preserved (W=0). Note that post-indexed addressing modes require explicit setting of the W bit, unlike LDR and STR which always write-back when post-indexed.

The value of the base register, modified by the offset in a pre-indexed instruction, is used as the address for the transfer of the first word. The second word (if more than one is transferred) will go to or come from an address one word (4 bytes) higher than the first transfer, and the address will be incremented by one word for each subsequent transfer.

ADDRESS ALIGNMENT

The base address should normally be a word aligned quantity. The bottom 2 bits of the address will appear on **A[1:0]** and might be interpreted by the memory system.

USE OF R15

If Rn is R15, the value used will be the address of the instruction plus 8 bytes. Base write-back to R15 must not be specified.

DATA ABORTS

If the address is legal but the memory manager generates an abort, the data trap will be taken. The write-back of the modified base will take place, but all other processor state will be preserved. The coprocessor is partly responsible for ensuring that the data transfer can be restarted after the cause of the abort has been resolved, and must ensure that any subsequent actions it undertakes can be repeated when the instruction is retried.

INSTRUCTION CYCLE TIMES

Coprocessor data transfer instructions take (n-1)S + 2N + bI incremental cycles to execute, where:

n	The number of words transferred.
b	The number of cycles spent in the coprocessor busy-wait loop.

S, N and I are defined as squential (S-cycle), non-squential (N-cycle), and internal (I-cycle), respectively.

ASSEMBLER SYNTAX

<LDC|STC>{cond}{L} p#,cd,<Address>

LDC	Load from memory to coprocessor
STC	Store from coprocessor to memory
{L}	When present perform long transfer (N=1), otherwise perform short transfer (N=0)
{cond}	Two character condition mnemonic. See Table 3-2..
p#	The unique number of the required coprocessor
cd	An expression evaluating to a valid coprocessor register number that is placed in the CRd field
<Address>	can be:
1	An expression which generates an address: The assembler will attempt to generate an instruction using the PC as a base and a corrected immediate offset to address the location given by evaluating the expression. This will be a PC relative, pre-indexed address. If the address is out of range, an error will be generated
2	A pre-indexed addressing specification: [Rn] offset of zero [Rn,<#expression>]{!} offset of <expression> bytes
3	A post-indexed addressing specification: [Rn],<#expression offset of <expression> bytes {!} write back the base register (set the W bit) if! is present Rn is an expression evaluating to a valid ARM7TDMI register number.

NOTE

If Rn is R15, the assembler will subtract 8 from the offset value to allow for ARM7TDMI pipelining.

EXAMPLES

```
        LDC     p1,c2,table     ;  Load c2 of coproc 1 from address
                                ;  table, using a PC relative address.
        STCEQL  p2,c3,[R5,#24]! ;  Conditionally store c3 of coproc 2
                                ;  into an address 24 bytes up from R5,
                                ;  write this address back to R5, and use
                                ;  long transfer option (probably to store multiple words).
```

NOTE

Although the address offset is expressed in bytes, the instruction offset field is in words. The assembler will adjust the offset appropriately.

COPROCESSOR REGISTER TRANSFERS (MRC, MCR)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2.. The instruction encoding is shown in Figure 3-27.

This class of instruction is used to communicate information directly between ARM7TDMI and a coprocessor. An example of a coprocessor to ARM7TDMI register transfer (MRC) instruction would be a FIX of a floating point value held in a coprocessor, where the floating point number is converted into a 32 bit integer within the coprocessor, and the result is then transferred to ARM7TDMI register. A FLOAT of a 32 bit value in ARM7TDMI register into a floating point value within the coprocessor illustrates the use of ARM7TDMI register to coprocessor transfer (MCR).

An important use of this instruction is to communicate control information directly from the coprocessor into the ARM7TDMI CPSR flags. As an example, the result of a comparison of two floating point values within a coprocessor can be moved to the CPSR to control the subsequent flow of execution.

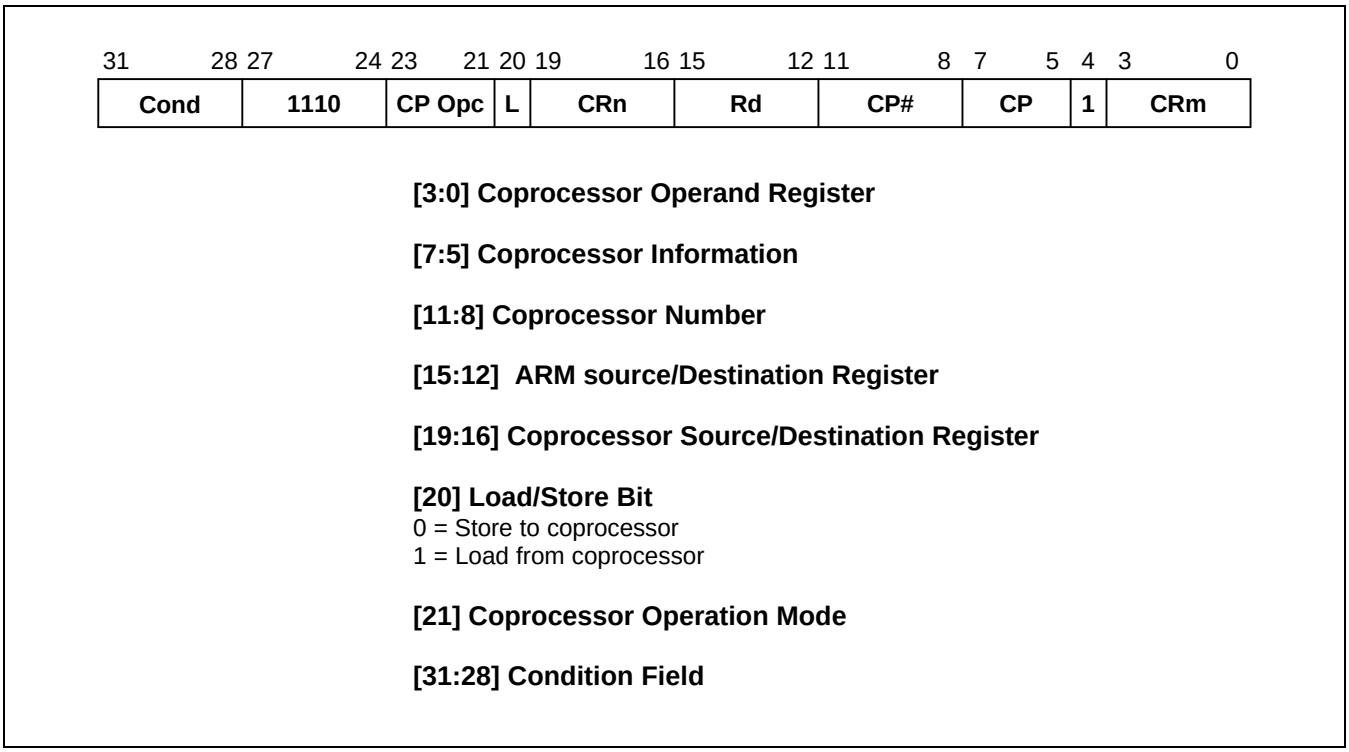


Figure 3-27. Coprocessor Register Transfer Instructions

THE COPROCESSOR FIELDS

The CP# field is used, as for all coprocessor instructions, to specify which coprocessor is being called upon.

The CP Opc, CRn, CP and CRm fields are used only by the coprocessor, and the interpretation presented here is derived from convention only. Other interpretations are allowed where the coprocessor functionality is incompatible with this one. The conventional interpretation is that the CP Opc and CP fields specify the operation the coprocessor is required to perform, CRn is the coprocessor register which is the source or destination of the transferred information, and CRm is a second coprocessor register which may be involved in some way which depends on the particular operation specified.

TRANSFERS TO R15

When a coprocessor register transfer to ARM7TDMI has R15 as the destination, bits 31, 30, 29 and 28 of the transferred word are copied into the N, Z, C and V flags respectively. The other bits of the transferred word are ignored, and the PC and other CPSR bits are unaffected by the transfer.

TRANSFERS FROM R15

A coprocessor register transfer from ARM7TDMI with R15 as the source register will store the PC+12.

INSTRUCTION CYCLE TIMES

MRC instructions take 1S + (b+1)I +1C incremental cycles to execute, where S, I and C are defined as squential (S-cycle), internal (I-cycle), and coprocessor register transfer (C-cycle), respectively. MCR instructions take 1S + bI +1C incremental cycles to execute, where *b* is the number of cycles spent in the coprocessor busy-wait loop.

ASSEMBLER SYNTAX

<MCR|MRC>{cond} p#,<expression1>,Rd,cn,cm{,<expression2>}

MRC	Move from coprocessor to ARM7TDMI register (L=1)
MCR	Move from ARM7TDMI register to coprocessor (L=0)
{cond}	Two character condition mnemonic. See Table 3-2
p#	The unique number of the required coprocessor
<expression1>	Evaluated to a constant and placed in the CP Opc field
Rd	An expression evaluating to a valid ARM7TDMI register number
cn and cm	Expressions evaluating to the valid coprocessor register numbers CRn and CRm respectively
<expression2>	Where present is evaluated to a constant and placed in the CP field

EXAMPLES

```
        MRC     p2,5,R3,c5,c6   ; Request coproc 2 to perform operation 5
                                ; on c5 and c6, and transfer the (single
                                ; 32-bit word) result back to R3.
        MCR     p6,0,R4,c5,c6   ; Request coproc 6 to perform operation 0
                                ; on R4 and place the result in c6.
        MRCEQ   p3,9,R3,c5,c6,2 ; Conditionally request coproc 3 to
                                ; perform operation 9 (type 2) on c5 and
                                ; c6, and transfer the result back to R3.
```

UNDEFINED INSTRUCTION

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction format is shown in Figure 3-28.

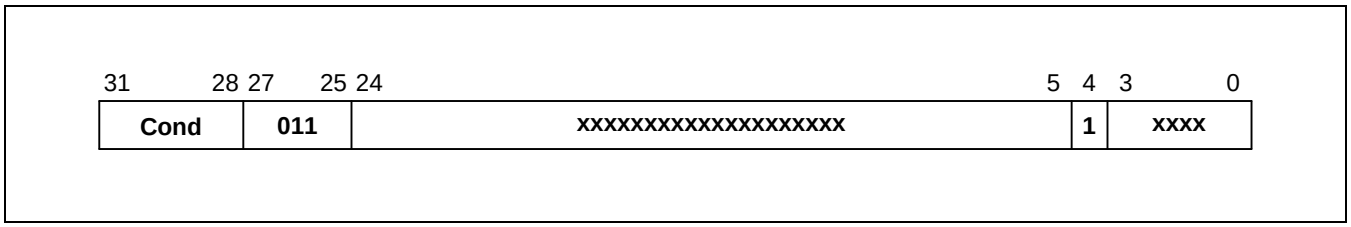


Figure 3-28. Undefined Instruction

If the condition is true, the undefined instruction trap will be taken.

Note that the undefined instruction mechanism involves offering this instruction to any coprocessors which may be present, and all coprocessors must refuse to accept it by driving **CPA** and **CPB** HIGH.

INSTRUCTION CYCLE TIMES

This instruction takes 2S + 1I + 1N cycles, where S, N and I are defined as squential (S-cycle), non-sequential (N-cycle), and internal (I-cycle).

ASSEMBLER SYNTAX

The assembler has no mnemonics for generating this instruction. If it is adopted in the future for some specified use, suitable mnemonics will be added to the assembler. Until such time, this instruction must not be used.

INSTRUCTION SET EXAMPLES

The following examples show ways in which the basic ARM7TDMI instructions can combine to give efficient code. None of these methods saves a great deal of execution time (although they may save some), mostly they just save code.

USING THE CONDITIONAL INSTRUCTIONS

Using Conditionals for Logical OR

```
CMP     Rn,#p          ;  If Rn=p OR Rm=q THEN GOTO Label.
BEQ     Label
CMP     Rm,#q
BEQ     Label
```

This can be replaced by

```
CMP     Rn,#p
CMPNE   Rm,#q          ;  If condition not satisfied try other test.
BEQ     Label
```

Absolute Value

```
TEQ     Rn,#0          ;  Test sign
RSBMI   Rn,Rn,#0       ;  and 2's complement if necessary.
```

Multiplication by 4, 5 or 6 (Run Time)

```
MOV     Rc,Ra,LSL#2    ;  Multiply by 4,
CMP     Rb,#5          ;  Test value,
ADDCS   Rc,Rc,Ra       ;  Complete multiply by 5,
ADDHI   Rc,Rc,Ra       ;  Complete multiply by 6.
```

Combining Discrete and Range Tests

```
TEQ     Rc,#127        ;  Discrete test,
CMPNE   Rc,#" "-1      ;  Range test
MOVLS   Rc,#"."        ;  IF Rc<= "" OR Rc=ASCII(127)
                       ;  THEN Rc:= "."
```

Division and Remainder

A number of divide routines for specific applications are provided in source form as part of the ANSI C library provided with the ARM Cross Development Toolkit, available from your supplier. A short general purpose divide routine follows.

```
                                    ;  Enter with numbers in Ra and Rb.
        MOV     Rcnt,#1             ;  Bit to control the division.
Div1    CMP     Rb,#0x80000000      ;  Move Rb until greater than Ra.
        CMPCC   Rb,Ra
        MOVCC   Rb,Rb,ASL#1
        MOVCC   Rcnt,Rcnt,ASL#1
        BCC     Div1
        MOV     Rc,#0
Div2    CMP     Ra,Rb               ;  Test for possible subtraction.
        SUBCS   Ra,Ra,Rb            ;  Subtract if ok,
        ADDCS   Rc,Rc,Rcnt          ;  Put relevant bit into result
        MOVS    Rcnt,Rcnt,LSR#1     ;  Shift control bit
        MOVNE   Rb,Rb,LSR#1         ;  Halve unless finished.
        BNE     Div2                ;  Divide result in Rc, remainder in Ra.
```

Overflow Eetection in the ARM7TDMI

1. Overflow in unsigned multiply with a 32-bit result

```
        UMULL   Rd,Rt,Rm,Rn         ;  3 to 6 cycles
        TEQ     Rt,#0               ;  +1 cycle and a register
        BNE     overflow
```

2. Overflow in signed multiply with a 32-bit result

```
        SMULL   Rd,Rt,Rm,Rn         ;  3 to 6 cycles
        TEQ     Rt,Rd ASR#31        ;  +1 cycle and a register
        BNE     overflow
```

3. Overflow in unsigned multiply accumulate with a 32 bit result

```
        UMLAL   Rd,Rt,Rm,Rn         ;  4 to 7 cycles
        TEQ     Rt,#0               ;  +1 cycle and a register
        BNE     overflow
```

4. Overflow in signed multiply accumulate with a 32 bit result

```
        SMLAL   Rd,Rt,Rm,Rn         ;  4 to 7 cycles
        TEQ     Rt,Rd, ASR#31       ;  +1 cycle and a register
        BNE     overflow
```

5. Overflow in unsigned multiply accumulate with a 64 bit result

```
UMULL   Rl,Rh,Rm,Rn     ;  3 to 6 cycles
ADDS    Rl,Rl,Ra1       ;  Lower accumulate
ADC     Rh,Rh,Ra2       ;  Upper accumulate
BCS     overflow        ;  1 cycle and 2 registers
```

6. Overflow in signed multiply accumulate with a 64 bit result

```
SMULL   Rl,Rh,Rm,Rn     ;  3 to 6 cycles
ADDS    Rl,Rl,Ra1       ;  Lower accumulate
ADC     Rh,Rh,Ra2       ;  Upper accumulate
BVS     overflow        ;  1 cycle and 2 registers
```

NOTE

Overflow checking is not applicable to unsigned and signed multiplies with a 64-bit result, since overflow does not occur in such calculations.

PSEUDO-RANDOM BINARY SEQUENCE GENERATOR

It is often necessary to generate (pseudo-) random numbers and the most efficient algorithms are based on shift generators with exclusive-OR feedback rather like a cyclic redundancy check generator. Unfortunately the sequence of a 32 bit generator needs more than one feedback tap to be maximal length (i.e. 2^32-1 cycles before repetition), so this example uses a 33 bit register with taps at bits 33 and 20. The basic algorithm is newbit:=bit 33 eor bit 20, shift left the 33 bit number and put in newbit at the bottom; this operation is performed for all the newbits needed (i.e. 32 bits). The entire operation can be done in 5 S cycles:

```
                        ;  Enter with seed in Ra (32 bits),
                        ;  Rb (1 bit in Rb lsb), uses Rc.
TST     Rb,Rb,LSR#1     ;  Top bit into carry
MOVS    Rc,Ra,RRX       ;  33 bit rotate right
ADC     Rb,Rb,Rb        ;  Carry into lsb of Rb
EOR     Rc,Rc,Ra,LSL#12 ;  (involved!)
EOR     Ra,Rc,Rc,LSR#20 ;  (similarly involved!) new seed in Ra, Rb as before
```

MULTIPLICATION BY CONSTANT USING THE BARREL SHIFTER

Multiplication by 2^n (1,2,4,8,16,32..)

```
MOV     Ra, Rb, LSL #n
```

Multiplication by 2^n+1 (3,5,9,17..)

```
ADD     Ra,Ra,Ra,LSL #n
```

Multiplication by 2^n-1 (3,7,15..)

```
RSB     Ra,Ra,Ra,LSL #n
```

Multiplication by 6

```
ADD     Ra,Ra,Ra,LSL #1     ;  Multiply by 3
MOV     Ra,Ra,LSL#1         ;  and then by 2
```

Multiply by 10 and add in extra number

```
ADD     Ra,Ra,Ra,LSL#2      ;  Multiply by 5
ADD     Ra,Rc,Ra,LSL#1      ;  Multiply by 2 and add in next digit
```

General recursive method for Rb := Ra*C, C a constant:

1. If C even, say C = 2^n*D, D odd:

```
D=1:    MOV   Rb,Ra,LSL #n
D<>1:   {Rb := Ra*D}
MOV     Rb,Rb,LSL #n
```

2. If C MOD 4 = 1, say C = 2^n*D+1, D odd, n>1:

```
D=1:    ADD   Rb,Ra,Ra,LSL #n
D<>1:   {Rb := Ra*D}
ADD     Rb,Ra,Rb,LSL #n
```

3. If C MOD 4 = 3, say C = 2^n*D-1, D odd, n>1:

```
D=1:    RSB   Rb,Ra,Ra,LSL #n
D<>1:   {Rb := Ra*D}
RSB     Rb,Ra,Rb,LSL #n
```

This is not quite optimal, but close. An example of its non-optimality is multiply by 45 which is done by:

```
RSB     Rb,Ra,Ra,LSL#2      ;  Multiply by 3
RSB     Rb,Ra,Rb,LSL#2      ;  Multiply by 4*3-1 = 11
ADD     Rb,Ra,Rb,LSL# 2     ;  Multiply by 4*11+1 = 45
```

rather than by:

```
ADD     Rb,Ra,Ra,LSL#3      ;  Multiply by 9
ADD     Rb,Rb,Rb,LSL#2      ;  Multiply by 5*9 = 45
```

LOADING A WORD FROM AN UNKNOWN ALIGNMENT

```
                                ; Enter with address in Ra (32 bits) uses
                                ; Rb, Rc result in Rd. Note d must be less than c e.g. 0,1
BIC     Rb,Ra,#3                ; Get word aligned address
LDMIA   Rb,{Rd,Rc}              ; Get 64 bits containing answer
AND     Rb,Ra,#3                ; Correction factor in bytes
MOVS    Rb,Rb,LSL#3             ;  ...now in bits and test if aligned
MOVNE   Rd,Rd,LSR Rb            ; Produce bottom of result word (if not aligned)
RSBNE   Rb,Rb,#32               ; Get other shift amount
ORRNE   Rd,Rd,Rc,LSL Rb         ; Combine two halves to get result
```

NOTES

THUMB INSTRUCTION SET FORMAT

The thumb instruction sets are 16-bit versions of ARM instruction sets (32-bit format). The ARM instructions are reduced to 16-bit versions, Thumb instructions, at the cost of versatile functions of the ARM instruction sets. The thumb instructions are decompressed to the ARM instructions by the Thumb decompressor inside the ARM7TDMI core.

As the Thumb instructions are compressed ARM instructions, the Thumb instructions have the 16-bit format instructions and have some restrictions. The restrictions by 16-bit format is fully notified for using the Thumb instructions.

FORMAT SUMMARY

The THUMB instruction set formats are shown in the following figure.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	0	0	0	Op		Offset5					Rs			Rd			Move Shifted register
2	0	0	0	1	1	I	Op	Rn/offset3			Rs			Rd			Add/subtract
3	0	0	1	Op		Rd			Offset8								Move/compare/add/ subtract immediate
4	0	1	0	0	0	0	Op				Rs			Rd			ALU operations
5	0	1	0	0	0	1	Op		H1	H2	Rs/Hs			Rd/Hd			Hi regiter operations /branch exchange
6	0	1	0	0	1	Rd			Word8								PC-relative load
7	0	1	0	1	L	B	0	Ro			Rb			Rd			Load/store with register offset
8	0	1	0	1	H	S	1	Ro			Rb			Rd			Load/store sign-extended byte/halfword
9	0	1	1	B	L	Offset5					Rb			Rd			Load/store with immediate offset
10	1	0	0	0	L	Offset5					Rb			Rd			Load/store halfword
11	1	0	0	1	L	Rd			Word8								SP-relative load/store
12	1	0	1	0	SP	Rd			Word8								Load address
13	1	0	1	1	0	0	0	0	S	SWord7							Add offset to stack pointer
14	1	0	1	1	L	1	0	R	Rlist								Push/pop register
15	1	1	0	0	L	Rb			Rlist								Multiple load/store
16	1	1	0	1	Cond				Softset8								Conditional branch
17	1	1	0	1	1	1	1	1	Value8								Software interrupt
18	1	1	1	0	0	Offset11											Unconditional branch
19	1	1	1	1	H	Offset											Long branch with link
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

Figure 3-29. THUMB Instruction Set Formats

OPCODE SUMMARY

The following table summarizes the THUMB instruction set. For further information about a particular instruction please refer to the sections listed in the right-most column.

Table 3-7. THUMB Instruction Set Opcodes

Mnemonic	Instruction	Lo-Register Operand	Hi-Register Operand	Condition Codes Set
ADC	Add with Carry	4	–	4
ADD	Add	4	–	4 (1)
AND	AND	4	–	4
ASR	Arithmetic Shift Right	4	–	4
B	Unconditional branch	4	–	–
Bxx	Conditional branch	4	–	–
BIC	Bit Clear	4	–	4
BL	Branch and Link	–	–	–
BX	Branch and Exchange	4	4	–
CMN	Compare Negative	4	–	4
CMP	Compare	4	4	4
EOR	EOR	4	–	4
LDMIA	Load multiple	4	–	–
LDR	Load word	4	–	–
LDRB	Load byte	4	–	–
LDRH	Load halfword	4	–	–
LSL	Logical Shift Left	4	–	4
LDSB	Load sign-extended byte	4	–	–
LDSH	Load sign-extended halfword	4	–	–
LSR	Logical Shift Right	4	–	4
MOV	Move register	4	4	4 (2)
MUL	Multiply	4	–	4
MVN	Move Negative register	4	–	4

Table 3-7. THUMB Instruction Set Opcodes (Continued)

Mnemonic	Instruction	Lo-Register Operand	Hi-Register Operand	Condition Codes Set
ADC	Add with Carry	4	–	4
ADD	Add	4	–	4 (1)
AND	AND	4	–	4
ASR	Arithmetic Shift Right	4	–	4
B	Unconditional branch	4	–	–
Bxx	Conditional branch	4	–	–
BIC	Bit Clear	4	–	4
BL	Branch and Link	–	–	–
BX	Branch and Exchange	4	4	–
CMN	Compare Negative	4	–	4
CMP	Compare	4	4	4
EOR	EOR	4	–	4
LDMIA	Load multiple	4	–	–

NOTES:

1. The condition codes are unaffected by the format 5, 12 and 13 versions of this instruction.
2. The condition codes are unaffected by the format 5 version of this instruction.

FORMAT 1: MOVE SHIFTED REGISTER

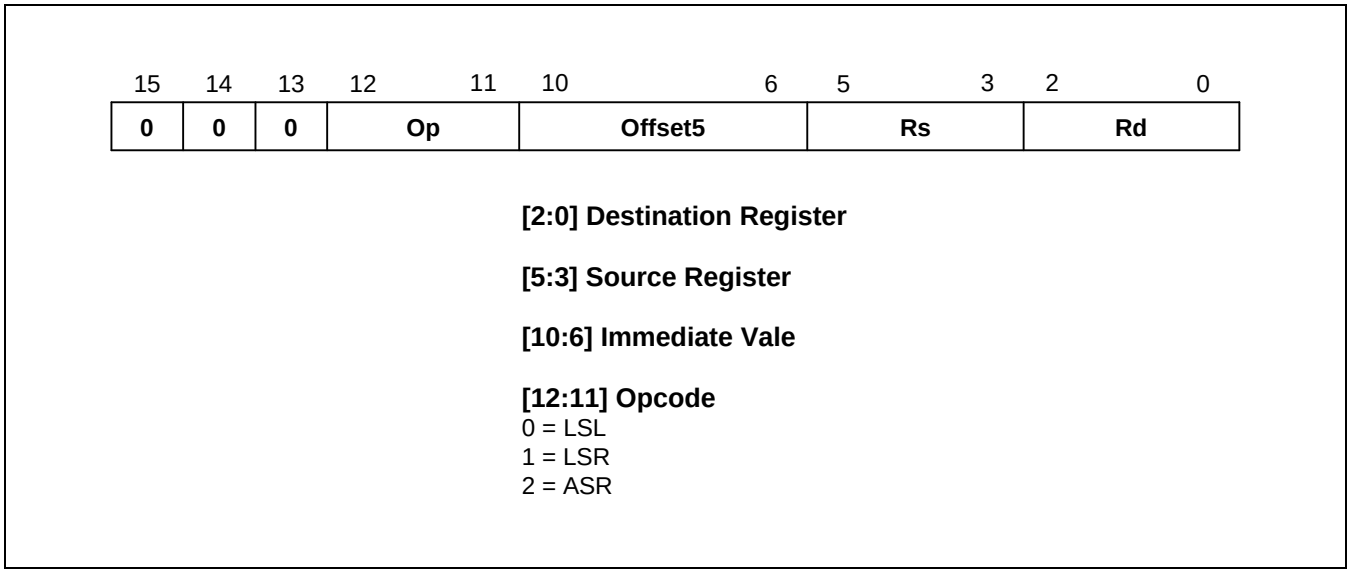


Figure 3-30. Format 1

OPERATION

These instructions move a shifted value between Lo registers. The THUMB assembler syntax is shown in Table 3-8.

NOTE

All instructions in this group set the CPSR condition codes.

Table 3-8. Summary of Format 1 Instructions

OP	THUMB Assembler	ARM Equipment	Action
00	LSL Rd, Rs, #Offset5	MOVS Rd, Rs, LSL #Offset5	Shift Rs left by a 5-bit immediate value and store the result in Rd.
01	LSR Rd, Rs, #Offset5	MOVS Rd, Rs, LSR #Offset5	Perform logical shift right on Rs by a 5-bit immediate value and store the result in Rd.
10	ASR Rd, Rs, #Offset5	MOVS Rd, Rs, ASR #Offset5	Perform arithmetic shift right on Rs by a 5-bit immediate value and store the result in Rd.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-8. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
LSR     R2, R5, #27     ; Logical shift right the contents
                        ; of R5 by 27 and store the result in R2.
                        ; Set condition codes on the result.
```

FORMAT 2: ADD/SUBTRACT

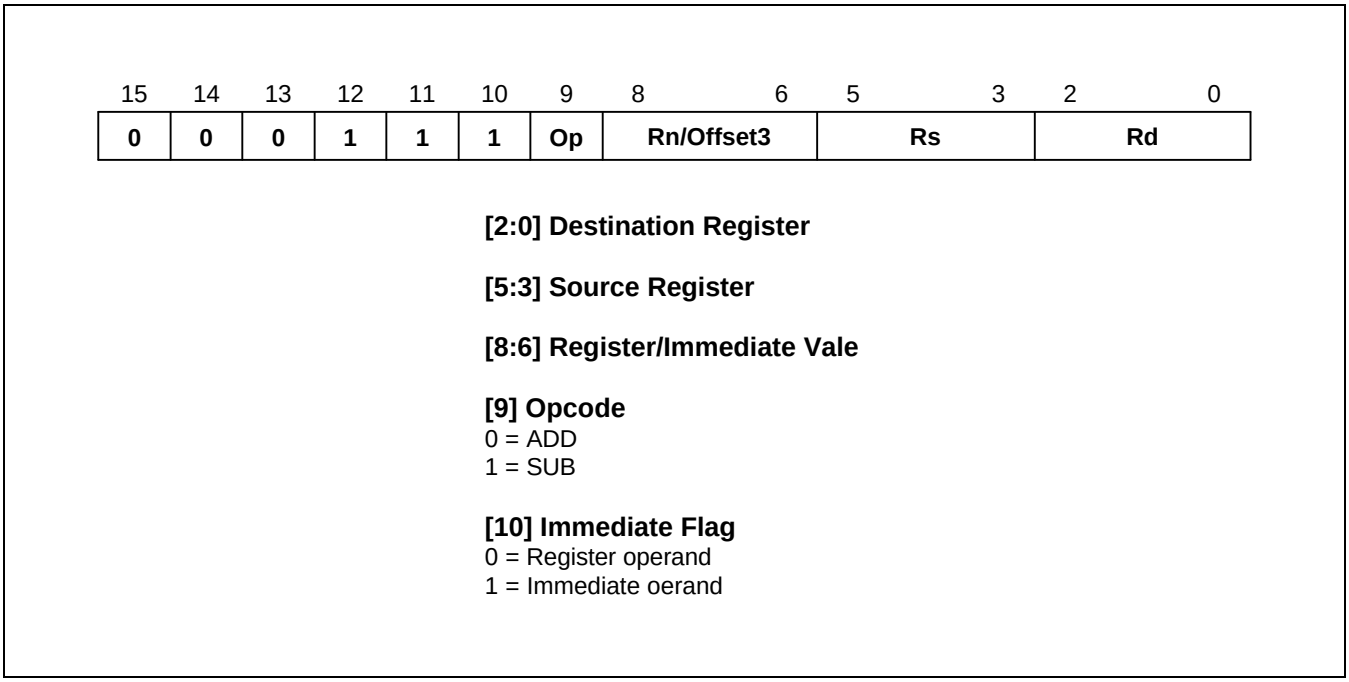


Figure 3-31. Format 2

OPERATION

These instructions allow the contents of a Lo register or a 3-bit immediate value to be added to or subtracted from a Lo register. The THUMB assembler syntax is shown in Table 3-9.

NOTE

All instructions in this group set the CPSR condition codes.

Table 3-9. Summary of Format 2 Instructions

OP	I	THUMB Assembler	ARM Equipment	Action
0	0	ADD Rd, Rs, Rn	ADDS Rd, Rs, Rn	Add contents of Rn to contents of Rs. Place result in Rd.
0	1	ADD Rd, Rs, #Offset3	ADDS Rd, Rs, #Offset3	Add 3-bit immediate value to contents of Rs. Place result in Rd.
1	0	SUB Rd, Rs, Rn	SUBS Rd, Rs, Rn	Subtract contents of Rn from contents of Rs. Place result in Rd.
1	1	SUB Rd, Rs, #Offset3	SUBS Rd, Rs, #Offset3	Subtract 3-bit immediate value from contents of Rs. Place result in Rd.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-9. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
ADD     R0, R3, R4      ;   R0 := R3 + R4 and set condition codes on the result.
SUB     R6, R2, #6      ;   R6 := R2 - 6 and set condition codes.
```

FORMAT 3: MOVE/COMPARE/ADD/SUBTRACT IMMEDIATE

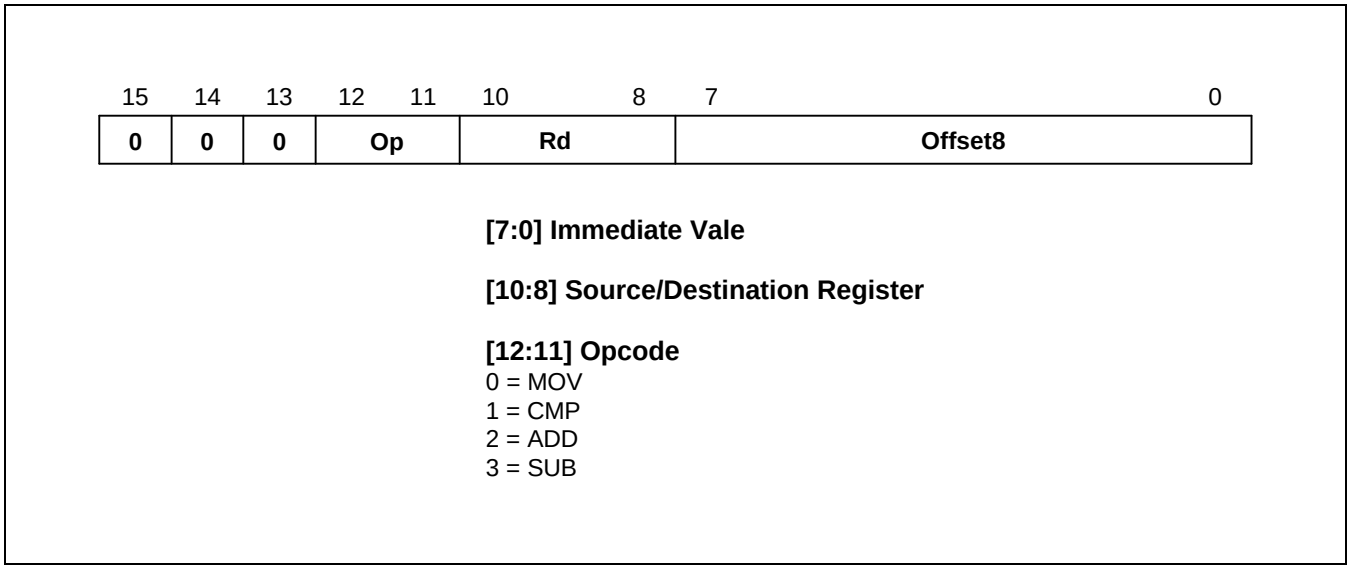


Figure 3-32. Format 3

OPERATIONS

The instructions in this group perform operations between a Lo register and an 8-bit immediate value. The THUMB assembler syntax is shown in Table 3-10.

NOTE

All instructions in this group set the CPSR condition codes.

Table 3-10. Summary of Format 3 Instructions

OP	THUMB Assembler	ARM Equipment	Action
00	MOV Rd, #Offset8	MOVS Rd, #Offset8	Move 8-bit immediate value into Rd.
01	CMP Rd, #Offset8	CMP Rd, #Offset8	Compare contents of Rd with 8-bit immediate value.
10	ADD Rd, #Offset8	ADDS Rd, Rd, #Offset8	Add 8-bit immediate value to contents of Rd and place the result in Rd.
11	SUB Rd, #Offset8	SUBS Rd, Rd, #Offset8	Subtract 8-bit immediate value from contents of Rd and place the result in Rd.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-10. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
MOV     R0, #128        ;   R0 := 128 and set condition codes
CMP     R2, #62         ;   Set condition codes on R2 - 62
ADD     R1, #255        ;   R1 := R1 + 255 and set condition codes
SUB     R6, #145        ;   R6 := R6 - 145 and set condition codes
```

FORMAT 4: ALU OPERATIONS

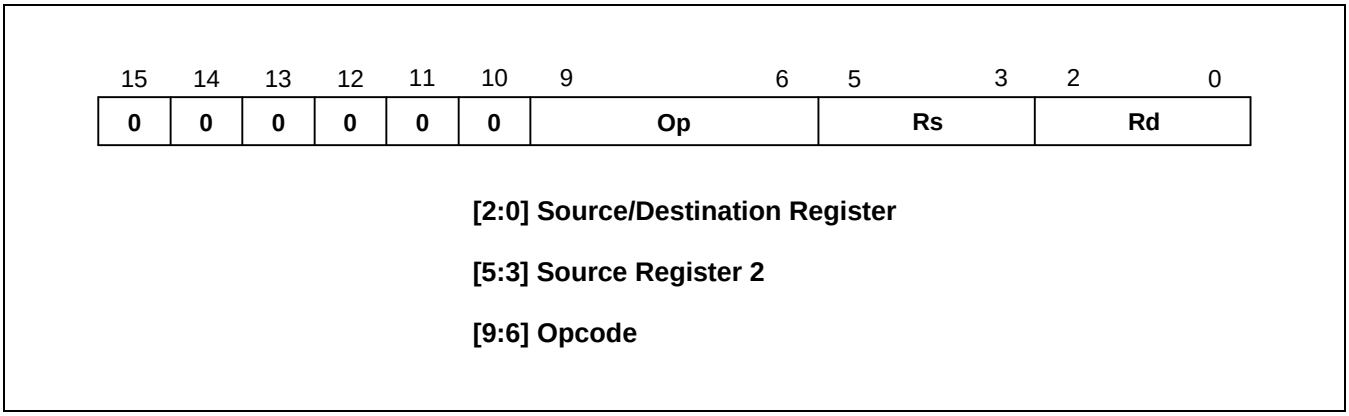


Figure 3-33. Format 4

OPERATION

The following instructions perform ALU operations on a Lo register pair.

NOTE

All instructions in this group set the CPSR condition codes.

Table 3-11. Summary of Format 4 Instructions

OP	THUMB Assembler	ARM Equipment	Action
0000	AND Rd, Rs	ANDS Rd, Rd, Rs	Rd:= Rd AND Rs
0001	EOR Rd, Rs	EORS Rd, Rd, Rs	Rd:= Rd EOR Rs
0010	LSL Rd, Rs	MOVS Rd, Rd, LSL Rs	Rd := Rd << Rs
0011	LSR Rd, Rs	MOVS Rd, Rd, LSR Rs	Rd := Rd >> Rs
0100	ASR Rd, Rs	MOVS Rd, Rd, ASR Rs	Rd := Rd ASR Rs
0101	ADC Rd, Rs	ADCS Rd, Rd, Rs	Rd := Rd + Rs + C-bit
0110	SBC Rd, Rs	SBCS Rd, Rd, Rs	Rd := Rd - Rs - NOT C-bit
0111	ROR Rd, Rs	MOVS Rd, Rd, ROR Rs	Rd := Rd ROR Rs
1000	TST Rd, Rs	TST Rd, Rs	Set condition codes on Rd AND Rs
1001	NEG Rd, Rs	RSBS Rd, Rs, #0	Rd = - Rs
1010	CMP Rd, Rs	CMP Rd, Rs	Set condition codes on Rd - Rs
1011	CMN Rd, Rs	CMN Rd, Rs	Set condition codes on Rd + Rs
1100	ORR Rd, Rs	ORRS Rd, Rd, Rs	Rd := Rd OR Rs
1101	MUL Rd, Rs	MULS Rd, Rs, Rd	Rd := Rs * Rd
1110	BIC Rd, Rs	BICS Rd, Rd, Rs	Rd := Rd AND NOT Rs
1111	MVN Rd, Rs	MVNS Rd, Rs	Rd := NOT Rs

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-11. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
EOR     R3, R4      ;  R3 := R3 EOR R4 and set condition codes
ROR     R1, R0      ;  Rotate Right R1 by the value in R0, store
                    ;  the result in R1 and set condition codes
NEG     R5, R3      ;  Subtract the contents of R3 from zero,
                    ;  Store the result in R5. Set condition codes ie R5 = - R3
CMP     R2, R6      ;  Set the condition codes on the result of R2 - R6
MUL     R0, R7      ;  R0 := R7 * R0 and set condition codes
```

FORMAT 5: HI-REGISTER OPERATIONS/BRANCH EXCHANGE

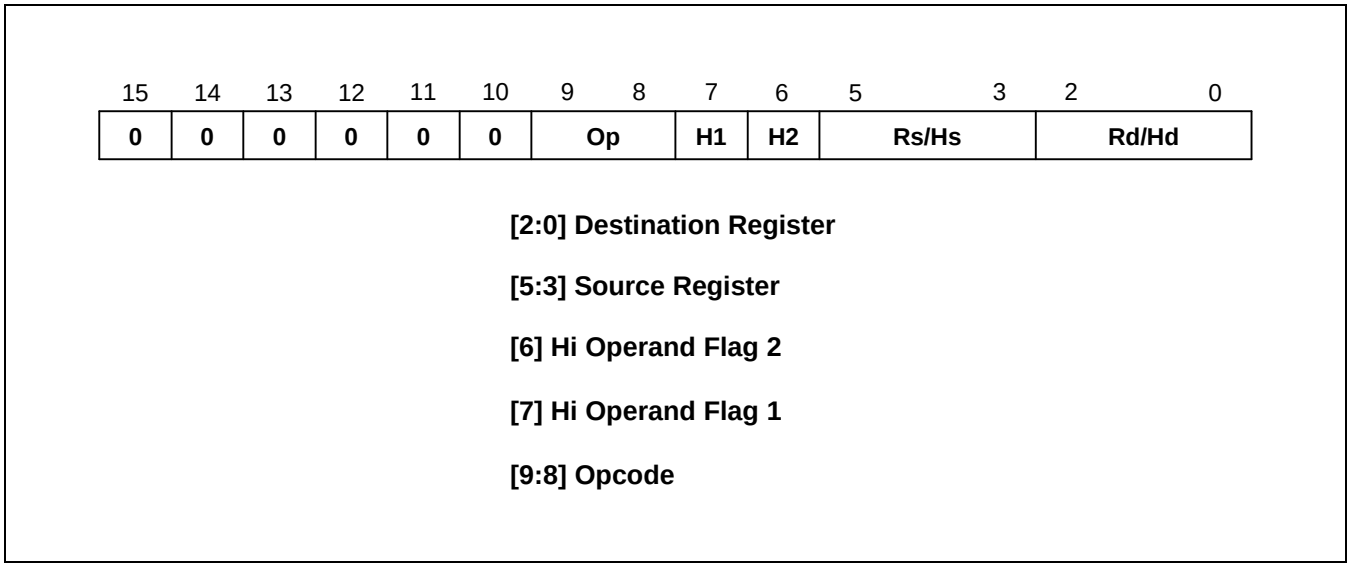


Figure 3-34. Format 5

OPERATION

There are four sets of instructions in this group. The first three allow ADD, CMP and MOV operations to be performed between Lo and Hi registers, or a pair of Hi registers. The fourth, BX, allows a Branch to be performed which may also be used to switch processor state. The THUMB assembler syntax is shown in Table 3-12.

NOTE

In this group only CMP (Op = 01) sets the CPSR condition codes.

The action of H1= 0, H2 = 0 for Op = 00 (ADD), Op =01 (CMP) and Op = 10 (MOV) is undefined, and should not be used.

Table 3-12. Summary of Format 5 Instructions

Op	H1	H2	THUMB assembler	ARM equivalent	Action
00	0	1	ADD Rd, Hs	ADD Rd, Rd, Hs	Add a register in the range 8-15 to a register in the range 0-7.
00	1	0	ADD Hd, Rs	ADD Hd, Hd, Rs	Add a register in the range 0-7 to a register in the range 8-15.
00	1	1	ADD Hd, Hs	ADD Hd, Hd, Hs	Add two registers in the range 8-15
01	0	1	CMP Rd, Hs	CMP Rd, Hs	Compare a register in the range 0-7 with a register in the range 8-15. Set the condition code flags on the result.
01	1	0	CMP Hd, Rs	CMP Hd, Rs	Compare a register in the range 8-15 with a register in the range 0-7. Set the condition code flags on the result.

Table 3-12. Summary of Format 5 Instructions (Continued)

Op	H1	H2	THUMB assembler	ARM equivalent	Action
01	1	1	CMP Hd, Hs	CMP Hd, Hs	Compare two registers in the range 8-15. Set the condition code flags on the result.
10	0	1	MOV Rd, Hs	MOV Rd, Hs	Move a value from a register in the range 8-15 to a register in the range 0-7.
10	1	0	MOV Hd, Rs	MOV Hd, Rs	Move a value from a register in the range 0-7 to a register in the range 8-15.
10	1	1	MOV Hd, Hs	MOV Hd, Hs	Move a value between two registers in the range 8-15.
11	0	0	BX Rs	BX Rs	Perform branch (plus optional state change) to address in a register in the range 0-7.
11	0	1	BX Hs	BX Hs	Perform branch (plus optional state change) to address in a register in the range 8-15.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-12. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

THE BX INSTRUCTION

BX performs a Branch to a routine whose start address is specified in a Lo or Hi register.

Bit 0 of the address determines the processor state on entry to the routine:

Bit 0 = 0 Causes the processor to enter ARM state.
Bit 0 = 1 Causes the processor to enter THUMB state.

NOTE

The action of H1 = 1 for this instruction is undefined, and should not be used.

EXAMPLES

Hi-Register Operations

```
ADD     PC, R5          ;  PC := PC + R5 but don't set the condition codes.
CMP     R4, R12         ;  Set the condition codes on the result of R4 - R12.
MOV     R15, R14        ;  Move R14 (LR) into R15 (PC)
                        ;  but don't set the condition codes,
                        ;  eg. return from subroutine.
```

Branch and Exchange

```
                        ;  Switch from THUMB to ARM state.
ADR     R1,outofTHUMB   ;  Load address of outofTHUMB into R1.
MOV     R11,R1
BX      R11             ;  Transfer the contents of R11 into the PC.
                        ;  Bit 0 of R11 determines whether
                        ;  ARM or THUMB state is entered, ie. ARM state here.
•
•
ALIGN
CODE32
outofTHUMB              ;  Now processing ARM instructions...
```

USING R15 AS AN OPERAND

If R15 is used as an operand, the value will be the address of the instruction + 4 with bit 0 cleared. Executing a BX PC in THUMB state from a non-word aligned address will result in unpredictable execution.

FORMAT 6: PC-RELATIVE LOAD

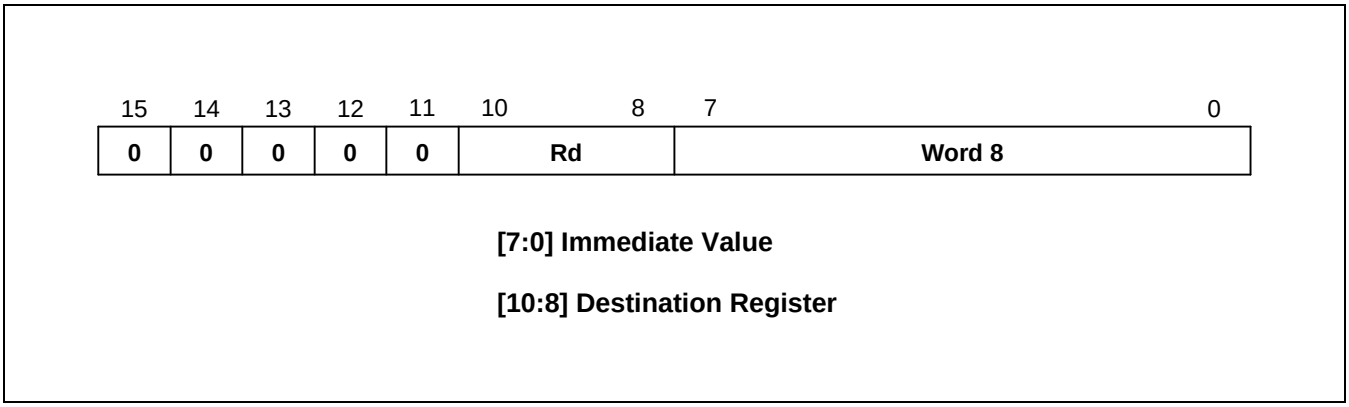


Figure 3-35. Format 6

OPERATION

This instruction loads a word from an address specified as a 10-bit immediate offset from the PC. The THUMB assembler syntax is shown below.

Table 3-13. Summary of PC-Relative Load Instruction

THUMB assembler	ARM equivalent	Action
LDR Rd, [PC, #Imm]	LDR Rd, [R15, #Imm]	Add unsigned offset (255 words, 1020 bytes) in Imm to the current value of the PC. Load the word from the resulting address into Rd.

NOTE: The value specified by #Imm is a full 10-bit address, but must always be word-aligned (ie with bits 1:0 set to 0), since the assembler places #Imm >> 2 in field Word 8. The value of the PC will be 4 bytes greater than the address of this instruction, but bit 1 of the PC is forced to 0 to ensure it is word aligned.

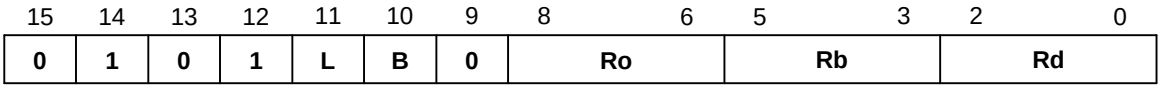
INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
LDR R3,[PC,#844]    ; Load into R3 the word found at the
                    ; address formed by adding 844 to PC.
                    ; bit[1] of PC is forced to zero.
                    ; Note that the THUMB opcode will contain
                    ; 211 as the Word8 value.
```

FORMAT 7: LOAD/STORE WITH REGISTER OFFSET



[2:0] Source/Destination Register

[5:3] Base Register

[8:6] Offset Register

[10] Byte/Word Flag
0 = Transfer word quantity
1 = Transfer byte quantity

[11] Load/Store Flag
0 = Store to memory
1 = Load from memory

Figure 3-36. Format 7

OPERATION

These instructions transfer byte or word values between registers and memory. Memory addresses are pre-indexed using an offset register in the range 0-7. The THUMB assembler syntax is shown in Table 3-14.

Table 3-14. Summary of Format 7 Instructions

L	B	THUMB assembler	ARM equivalent	Action
0	0	STR Rd, [Rb, Ro]	STR Rd, [Rb, Ro]	Pre-indexed word store: Calculate the target address by adding together the value in Rb and the value in Ro. Store the contents of Rd at the address.
0	1	STRB Rd, [Rb, Ro]	STRB Rd, [Rb, Ro]	Pre-indexed byte store: Calculate the target address by adding together the value in Rb and the value in Ro. Store the byte value in Rd at the resulting address.
1	0	LDR Rd, [Rb, Ro]	LDR Rd, [Rb, Ro]	Pre-indexed word load: Calculate the source address by adding together the value in Rb and the value in Ro. Load the contents of the address into Rd.
1	1	LDRB Rd, [Rb, Ro]	LDRB Rd, [Rb, Ro]	Pre-indexed byte load: Calculate the source address by adding together the value in Rb and the value in Ro. Load the byte value at the resulting address.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-14. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
STR     R3, [R2,R6]     ; Store word in R3 at the address
                        ; formed by adding R6 to R2.
LDRB    R2, [R0,R7]     ; Load into R2 the byte found at
                        ; the address formed by adding R7 to R0.
```

FORMAT 8: LOAD/STORE SIGN-EXTENDED BYTE/HALFWORD

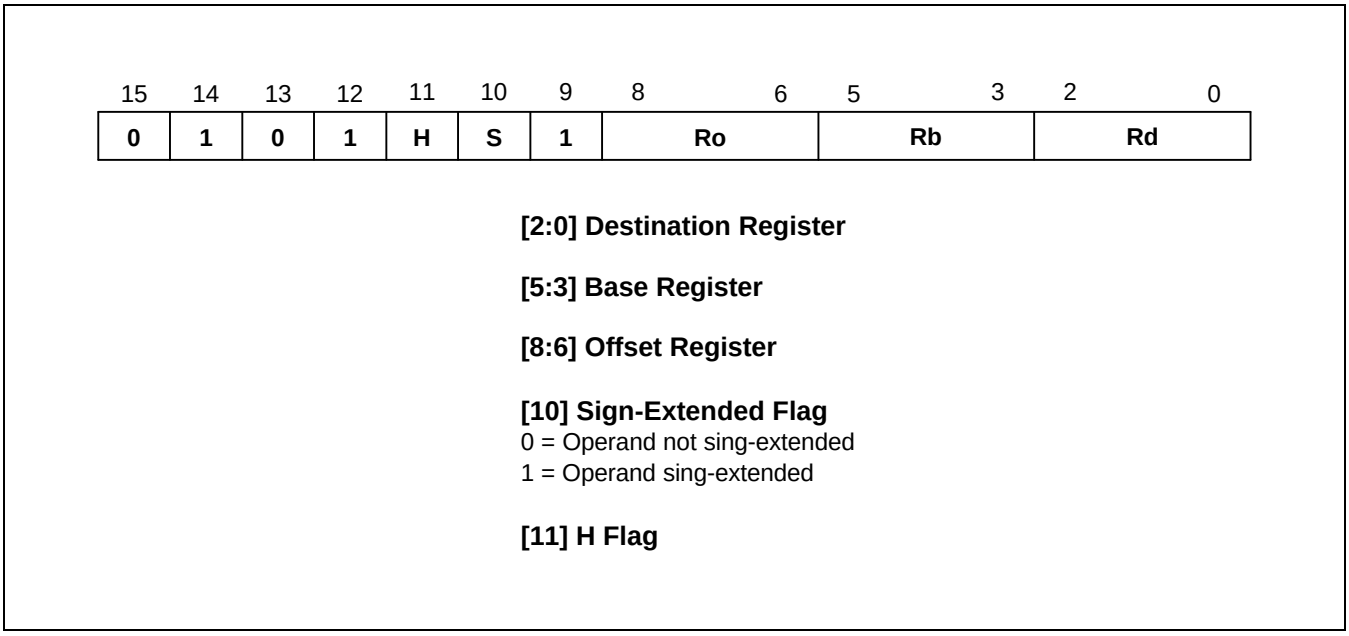


Figure 3-37. Format 8

OPERATION

These instructions load optionally sign-extended bytes or halfwords, and store halfwords. The THUMB assembler syntax is shown below.

Table 3-15. Summary of format 8 instructions

L	B	THUMB assembler	ARM equivalent	Action
0	0	STRH Rd, [Rb, Ro]	STRH Rd, [Rb, Ro]	Store halfword: Add Ro to base address in Rb. Store bits 0-15 of Rd at the resulting address.
0	1	LDRH Rd, [Rb, Ro]	LDRH Rd, [Rb, Ro]	Load halfword: Add Ro to base address in Rb. Load bits 0-15 of Rd from the resulting address, and set bits 16-31 of Rd to 0.
1	0	LDSB Rd, [Rb, Ro]	LDRSB Rd, [Rb, Ro]	Load sign-extended byte: Add Ro to base address in Rb. Load bits 0-7 of Rd from the resulting address, and set bits 8-31 of Rd to bit 7.
1	1	LDSH Rd, [Rb, Ro]	LDRSH Rd, [Rb, Ro]	Load sign-extended halfword: Add Ro to base address in Rb. Load bits 0-15 of Rd from the resulting address, and set bits 16-31 of Rd to bit 15.

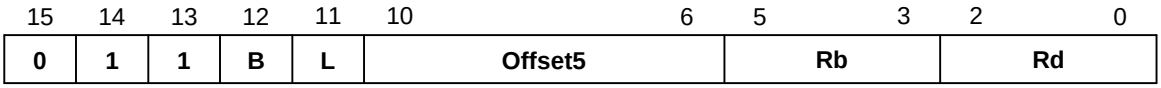
INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-15. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
STRH    R4, [R3, R0]    ;  Store the lower 16 bits of R4 at the
                        ;  address formed by adding R0 to R3.
LDSB    R2, [R7, R1]    ;  Load into R2 the sign extended byte
                        ;  found at the address formed by adding R1 to R7.
LDSH    R3, [R4, R2]    ;  Load into R3 the sign extended halfword
                        ;  found at the address formed by adding R2 to R4.
```

FORMAT 9: LOAD/STORE WITH IMMEDIATE OFFSET



[2:0] Source/Destination Register

[5:3] Base Register

[10:6] Offset Register

[11] Load/Store Flag
0 = Store to memory
1 = Load from memory

[12] Byte/Word Flad
0 = Transfer word quantity
1 = Transfer byte quantity

Figure 3-38. Format 9

OPERATION

These instructions transfer byte or word values between registers and memory using an immediate 5 or 7-bit offset. The THUMB assembler syntax is shown in Table 3-16.

Table 3-16. Summary of Format 9 Instructions

L	B	THUMB assembler	ARM equivalent	Action
0	0	STR Rd, [Rb, #Imm]	STR Rd, [Rb, #Imm]	Calculate the target address by adding together the value in Rb and Imm. Store the contents of Rd at the address.
1	0	LDR Rd, [Rb, #Imm]	LDR Rd, [Rb, #Imm]	Calculate the source address by adding together the value in Rb and Imm. Load Rd from the address.
0	1	STRB Rd, [Rb, #Imm]	STRB Rd, [Rb, #Imm]	Calculate the target address by adding together the value in Rb and Imm. Store the byte value in Rd at the address.
1	1	LDRB Rd, [Rb, #Imm]	LDRB Rd, [Rb, #Imm]	Calculate source address by adding together the value in Rb and Imm. Load the byte value at the address into Rd.

NOTE: For word accesses (B = 0), the value specified by #Imm is a full 7-bit address, but must be word-aligned (ie with bits 1:0 set to 0), since the assembler places #Imm >> 2 in the Offset5 field.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-16. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
LDR     R2, [R5,#116]   ; Load into R2 the word found at the
                        ; address formed by adding 116 to R5.
                        ; Note that the THUMB opcode will
                        ; contain 29 as the Offset5 value.
STRB    R1, [R0,#13]    ; Store the lower 8 bits of R1 at the
                        ; address formed by adding 13 to R0.
                        ; Note that the THUMB opcode will
                        ; contain 13 as the Offset5 value.
```

FORMAT 10: LOAD/STORE HALFWORD

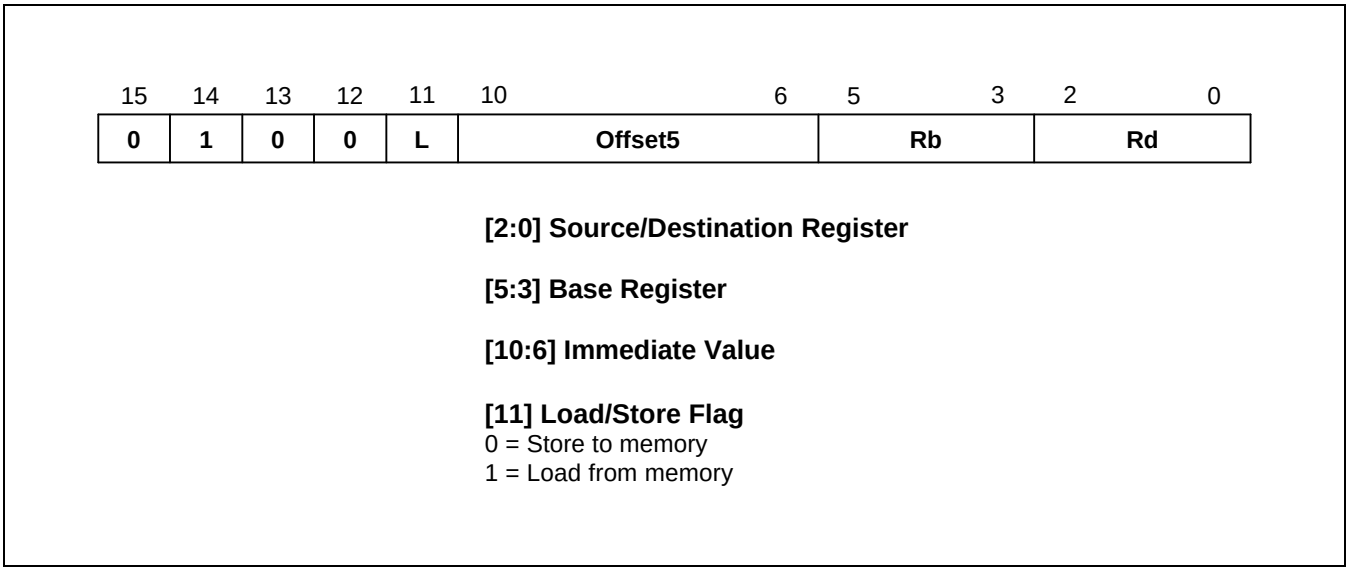


Figure 3-39. Format 10

OPERATION

These instructions transfer halfword values between a Lo register and memory. Addresses are pre-indexed, using a 6-bit immediate value. The THUMB assembler syntax is shown in Table 3-17.

Table 3-17. Halfword Data Transfer Instructions

L	THUMB assembler	ARM equivalent	Action
0	STRH Rd, [Rb, #Imm]	STRH Rd, [Rb, #Imm]	Add #Imm to base address in Rb and store bits 0–15 of Rd at the resulting address.
1	LDRH Rd, [Rb, #Imm]	LDRH Rd, [Rb, #Imm]	Add #Imm to base address in Rb. Load bits 0-15 from the resulting address into Rd and set bits 16-31 to zero.

NOTE: #Imm is a full 6-bit address but must be halfword-aligned (ie with bit 0 set to 0) since the assembler places #Imm >> 1 in the Offset5 field.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-17. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
STRH    R6, [R1, #56]   ; Store the lower 16 bits of R4 at the address formed by
                        ; adding 56 R1. Note that the THUMB opcode will contain
                        ; 28 as the Offset5 value.
LDRH    R4, [R7, #4]    ; Load into R4 the halfword found at the address formed by
                        ; adding 4 to R7. Note that the THUMB opcode will contain
                        ; 2 as the Offset5 value.
```

FORMAT 11: SP-RELATIVE LOAD/STORE

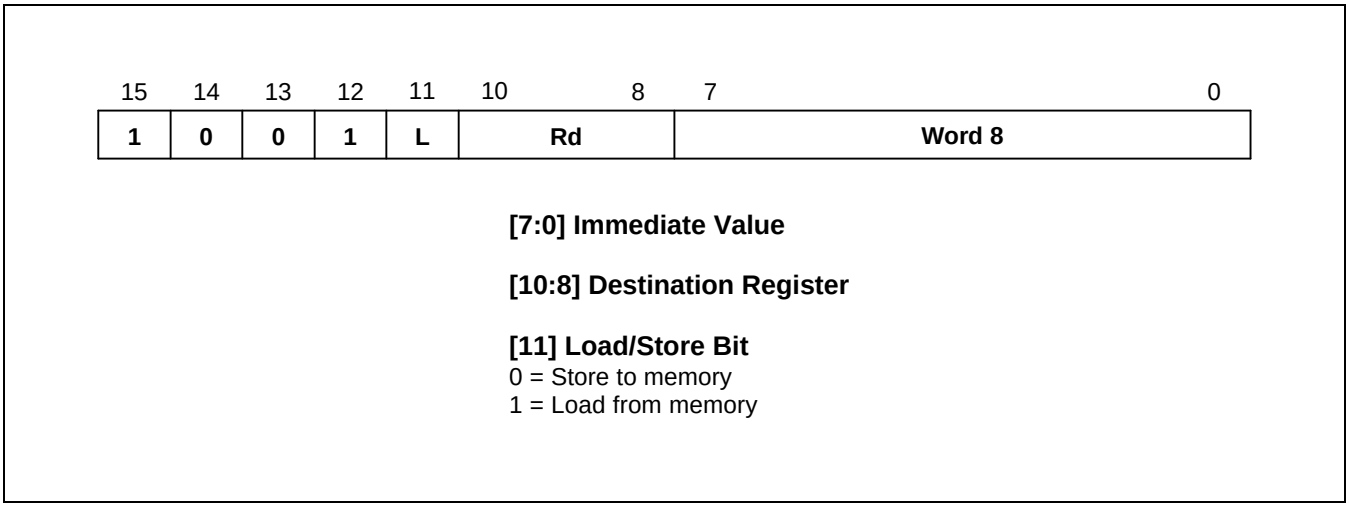


Figure 3-40. Format 11

OPERATION

The instructions in this group perform an SP-relative load or store.The THUMB assembler syntax is shown in the following table.

Table 3-18. SP-Relative Load/Store Instructions

L	THUMB assembler	ARM equivalent	Action
0	STR Rd, [SP, #Imm]	STR Rd, [R13 #Imm]	Add unsigned offset (255 words, 1020 bytes) in Imm to the current value of the SP (R7). Store the contents of Rd at the resulting address.
1	LDR Rd, [SP, #Imm]	LDR Rd, [R13 #Imm]	Add unsigned offset (255 words, 1020 bytes) in Imm to the current value of the SP (R7). Load the word from the resulting address into Rd.

NOTE: The offset supplied in #Imm is a full 10-bit address, but must always be word-aligned (ie bits 1:0 set to 0), since the assembler places #Imm >> 2 in the Word8 field.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-18. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
STR     R4, [SP,#492]   ; Store the contents of R4 at the address
                        ; formed by adding 492 to SP (R13).
                        ; Note that the THUMB opcode will contain
                        ; 123 as the Word8 value.
```

FORMAT 12: LOAD ADDRESS

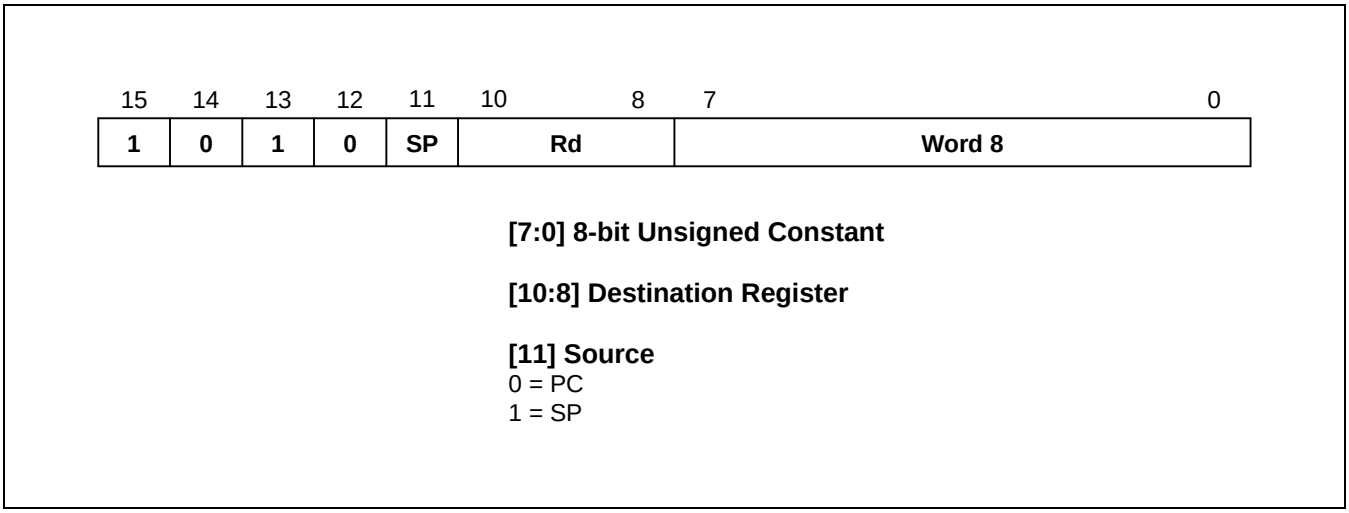


Figure 3-41. Format 12

OPERATION

These instructions calculate an address by adding an 10-bit constant to either the PC or the SP, and load the resulting address into a register. The THUMB assembler syntax is shown in the following table.

Table 3-19. Load Address

L	THUMB assembler	ARM equivalent	Action
0	ADD Rd, PC, #Imm	ADD Rd, R15, #Imm	Add #Imm to the current value of the program counter (PC) and load the result into Rd.
1	ADD Rd, SP, #Imm	ADD Rd, R13, #Imm	Add #Imm to the current value of the stack pointer (SP) and load the result into Rd.

NOTE: The value specified by #Imm is a full 10-bit value, but this must be word-aligned (ie with bits 1:0 set to 0) since the assembler places #Imm >> 2 in field Word 8.

Where the PC is used as the source register (SP = 0), bit 1 of the PC is always read as 0. The value of the PC will be 4 bytes greater than the address of the instruction before bit 1 is forced to 0.

The CPSR condition codes are unaffected by these instructions.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-19. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
ADD     R2, PC, #572    ; R2 := PC + 572, but don't set the
                        ; condition codes. bit[1] of PC is forced to zero.
                        ; Note that the THUMB opcode will
                        ; contain 143 as the Word8 value.
ADD     R6, SP, #212    ; R6 := SP (R13) + 212, but don't
                        ; set the condition codes.
                        ; Note that the THUMB opcode will
                        ; contain 53 as the Word 8 value.
```

FORMAT 13: ADD OFFSET TO STACK POINTER

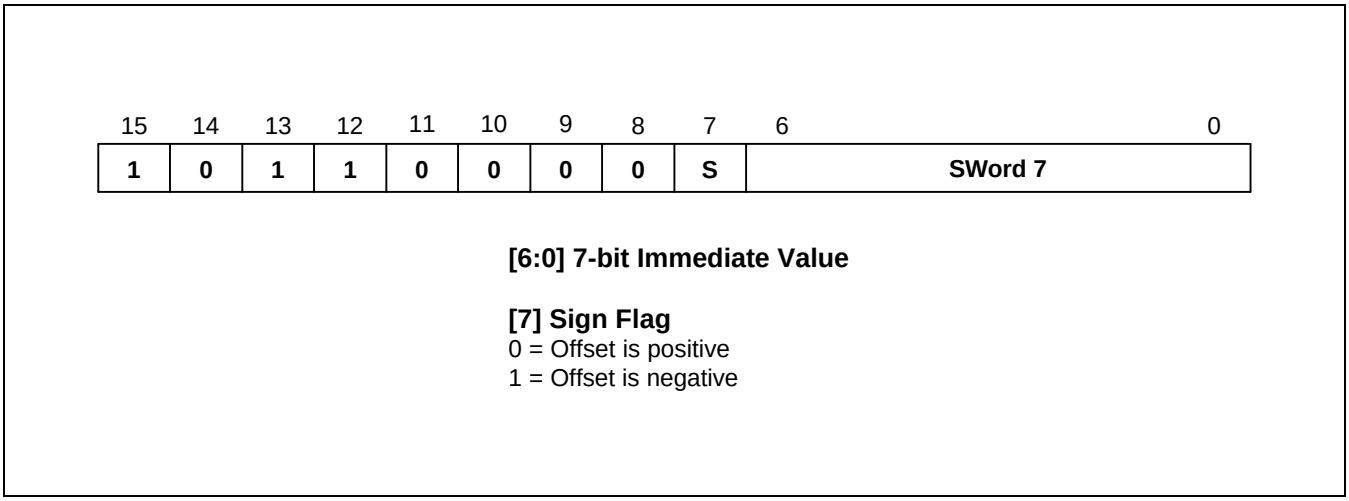


Figure 3-42. Format 13

OPERATION

This instruction adds a 9-bit signed constant to the stack pointer. The following table shows the THUMB assembler syntax.

Table 3-20. The ADD SP Instruction

L	THUMB assembler	ARM equivalent	Action
0	ADD SP, #Imm	ADD R13, R13, #Imm	Add #Imm to the stack pointer (SP).
1	ADD SP, # -Imm	SUB R13, R13, #Imm	Add #-Imm to the stack pointer (SP).

NOTE: The offset specified by #Imm can be up to -/+ 508, but must be word-aligned (ie with bits 1:0 set to 0) since the assembler converts #Imm to an 8-bit sign + magnitude number before placing it in field SWord7. The condition codes are not set by this instruction.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-20. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
ADD     SP, #268        ;  SP (R13) := SP + 268, but don't set the condition codes.
                        ;  Note that the THUMB opcode will
                        ;  contain 67 as the Word7 value and S=0.
ADD     SP, #-104       ;  SP (R13) := SP - 104, but don't set the condition codes.
                        ;  Note that the THUMB opcode will contain
                        ;  26 as the Word7 value and S=1.
```

FORMAT 14: PUSH/POP REGISTERS

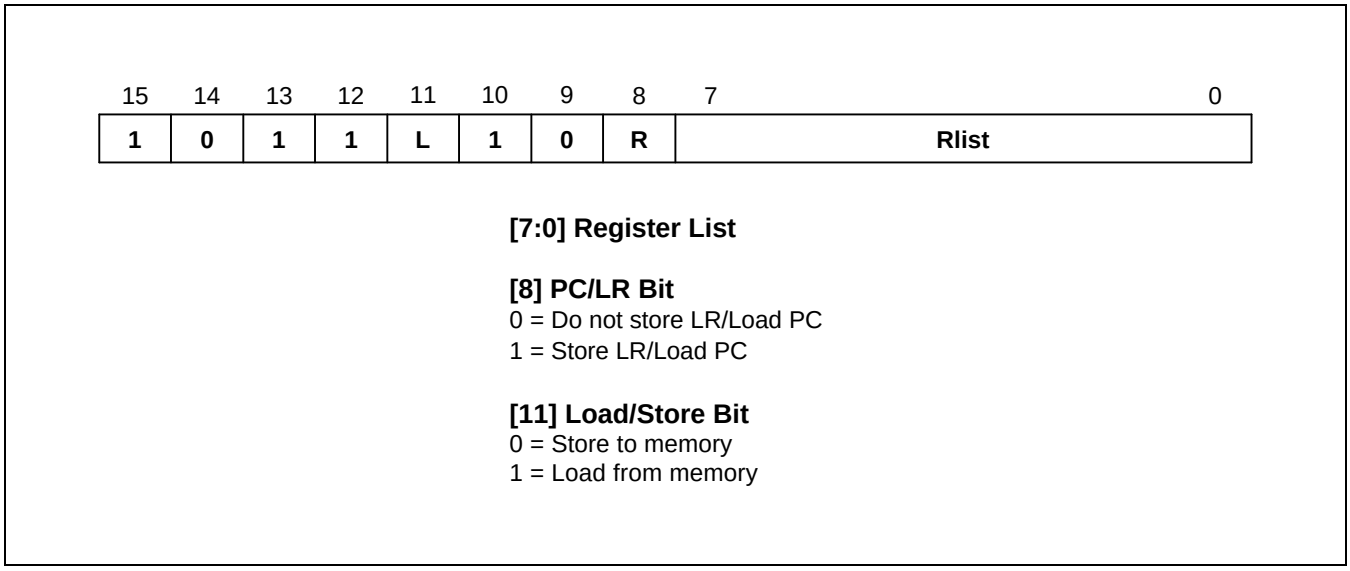


Figure 3-43. Format 14

OPERATION

The instructions in this group allow registers 0-7 and optionally LR to be pushed onto the stack, and registers 0-7 and optionally PC to be popped off the stack. The THUMB assembler syntax is shown in Table 3-21.

NOTE

The stack is always assumed to be Full Descending.

Table 3-21. PUSH and POP Instructions

L	B	THUMB assembler	ARM equivalent	Action
0	0	PUSH { Rlist }	STMDB R13!, { Rlist }	Push the registers specified by Rlist onto the stack. Update the stack pointer.
0	1	PUSH { Rlist, LR }	STMDB R13!, { Rlist, R14 }	Push the Link Register and the registers specified by Rlist (if any) onto the stack. Update the stack pointer.
1	0	POP { Rlist }	LDMIA R13!, { Rlist }	Pop values off the stack into the registers specified by Rlist. Update the stack pointer.
1	1	POP { Rlist, PC }	LDMIA R13!, {Rlist, R15}	Pop values off the stack and load into the registers specified by Rlist. Pop the PC off the stack. Update the stack pointer.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-21. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
PUSH    {R0-R4,LR}      ; Store R0,R1,R2,R3,R4 and R14 (LR) at
                        ; the stack pointed to by R13 (SP) and update R13.
                        ; Useful at start of a sub-routine to
                        ; save workspace and return address.
POP     {R2,R6,PC}      ; Load R2,R6 and R15 (PC) from the stack
                        ; pointed to by R13 (SP) and update R13.
                        ; Useful to restore workspace and return from sub-routine.
```

FORMAT 15: MULTIPLE LOAD/STORE

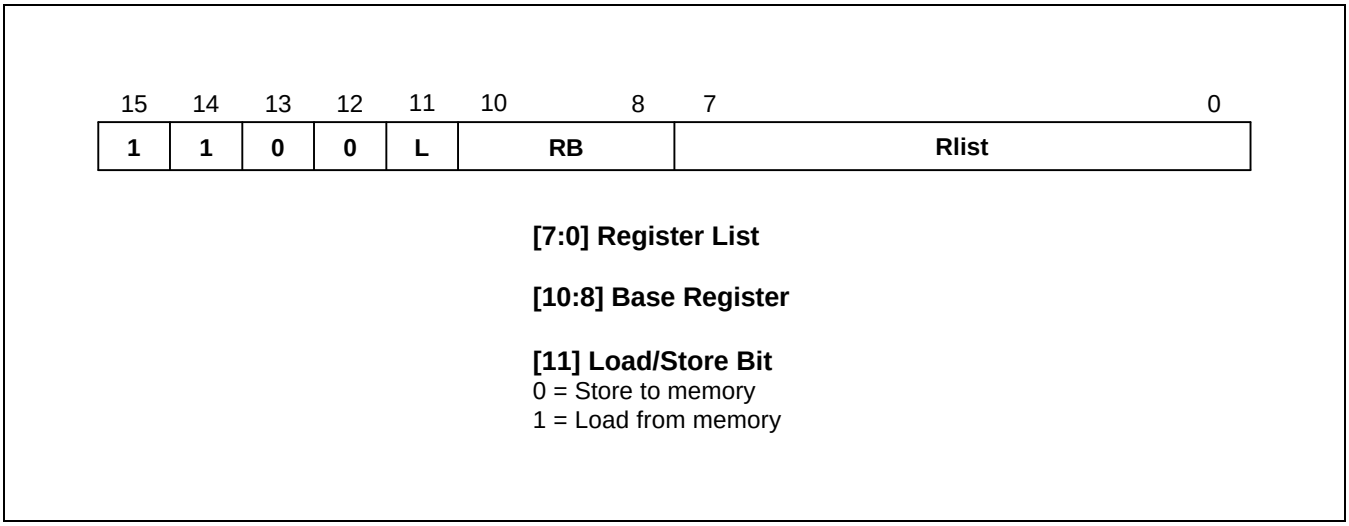


Figure 3-44. Format 15

OPERATION

These instructions allow multiple loading and storing of Lo registers. The THUMB assembler syntax is shown in the following table.

Table 3-22. The Multiple Load/Store Instructions

L	THUMB assembler	ARM equivalent	Action
0	STMIA Rb!, { Rlist }	STMIA Rb!, { Rlist }	Store the registers specified by Rlist, starting at the base address in Rb. Write back the new base address.
1	LDMIA Rb!, { Rlist }	LDMIA Rb!, { Rlist }	Load the registers specified by Rlist, starting at the base address in Rb. Write back the new base address.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-22. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
STMIA    R0!, {R3-R7}      ; Store the contents of registers R3-R7
                           ; starting at the address specified in
                           ; R0, incrementing the addresses for each word.
                           ; Write back the updated value of R0.
```

FORMAT 16: CONDITIONAL BRANCH

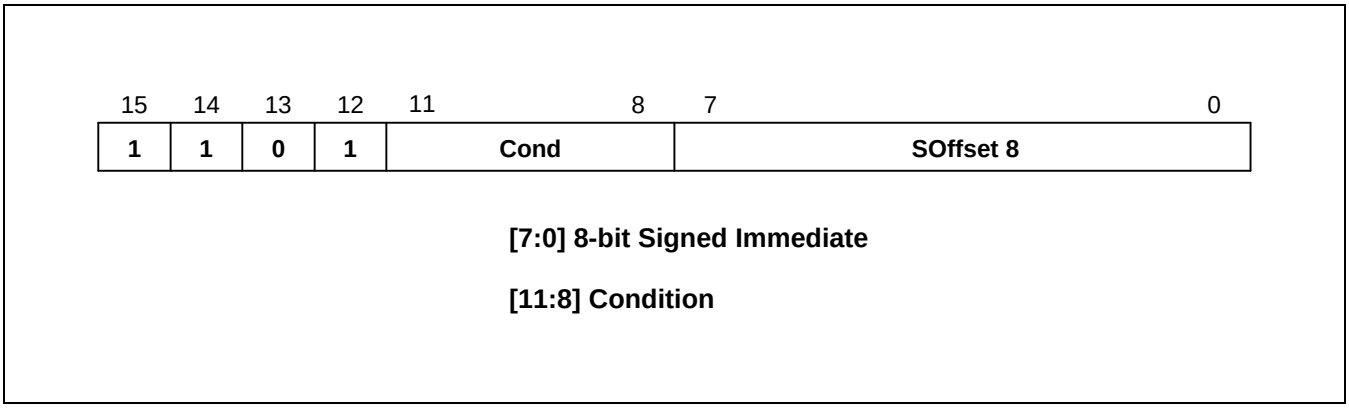


Figure 3-45. Format 16

OPERATION

The instructions in this group all perform a conditional Branch depending on the state of the CPSR condition codes. The branch offset must take account of the prefetch operation, which causes the PC to be 1 word (4 bytes) ahead of the current instruction.

The THUMB assembler syntax is shown in the following table.

Table 2-23. The Conditional Branch Instructions

L	THUMB assembler	ARM equivalent	Action
0000	BEQ label	BEQ label	Branch if Z set (equal)
0001	BNE label	BNE label	Branch if Z clear (not equal)
0010	BCS label	BCS label	Branch if C set (unsigned higher or same)
0011	BCC label	BCC label	Branch if C clear (unsigned lower)
0100	BMI label	BMI label	Branch if N set (negative)
0101	BPL label	BPL label	Branch if N clear (positive or zero)
0110	BVS label	BVS label	Branch if V set (overflow)
0111	BVC label	BVC label	Branch if V clear (no overflow)
1000	BHI label	BHI label	Branch if C set and Z clear (unsigned higher)

Table 2-23. The Conditional Branch Instructions (Continued)

L	THUMB assembler	ARM equivalent	Action
1001	BLS label	BLS label	Branch if C clear or Z set (unsigned lower or same)
1010	BGE label	BGE label	Branch if N set and V set, or N clear and V clear (greater or equal)
1011	BLT label	BLT label	Branch if N set and V clear, or N clear and V set (less than)
1100	BGT label	BGT label	Branch if Z clear, and either N set and V set or N clear and V clear (greater than)
1101	BLE label	BLE label	Branch if Z set, or N set and V clear, or N clear and V set (less than or equal)

NOTES

1. While label specifies a full 9-bit two's complement address, this must always be halfword-aligned (ie with bit 0 set to 0) since the assembler actually places label >> 1 in field SOffset8.
2. Cond = 1110 is undefined, and should not be used.
   Cond = 1111 creates the SWI instruction: see .

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-23. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
        CMP R0, #45          ;  Branch to 'over' if R0 > 45.
        BGT over             ;  Note that the THUMB opcode will contain
        •                    ;  the number of halfwords to offset.
        •
over    •                    ;  Must be halfword aligned.
```

FORMAT 17: SOFTWARE INTERRUPT

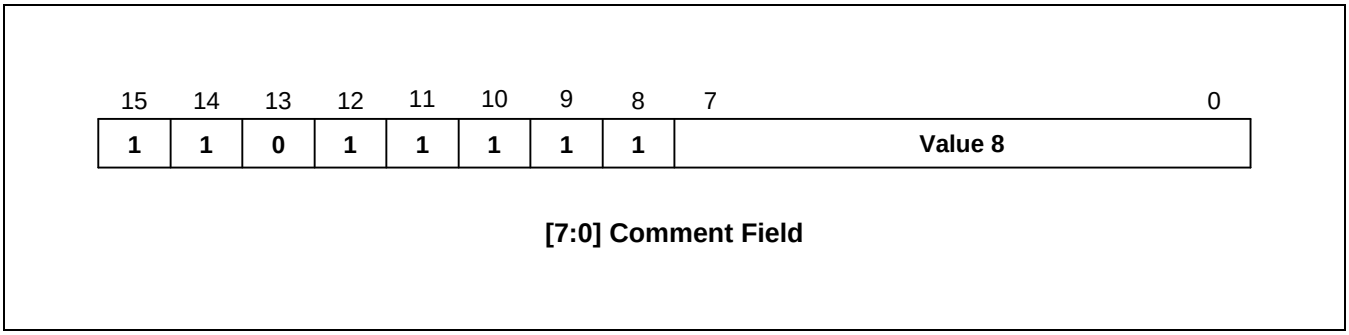


Figure 3-46. Format 17

OPERATION

The SWI instruction performs a software interrupt. On taking the SWI, the processor switches into ARM state and enters Supervisor (SVC) mode.

The THUMB assembler syntax for this instruction is shown below.

Table 3-24. The SWI Instruction

THUMB assembler	ARM equivalent	Action
SWI Value 8	SWI Value 8	Perform Software Interrupt: Move the address of the next instruction into LR, move CPSR to SPSR, load the SWI vector address (0x8) into the PC. Switch to ARM state and enter SVC mode.

NOTE: Value8 is used solely by the SWI handler; it is ignored by the processor.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-24. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
        SWI 18          ; Take the software interrupt exception.
                        ; Enter Supervisor mode with 18 as the
                        ; requested SWI number.
```

FORMAT 18: UNCONDITIONAL BRANCH

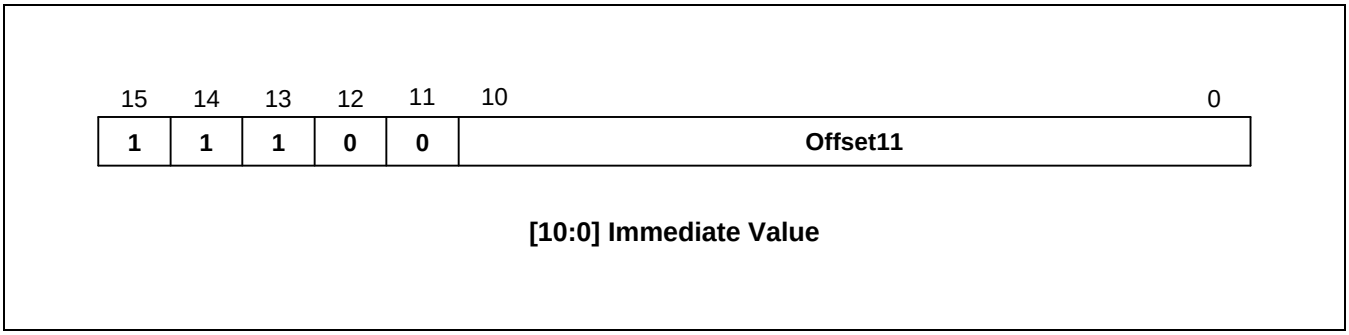


Figure 3-47. Format 18

OPERATION

This instruction performs a PC-relative Branch. The THUMB assembler syntax is shown below. The branch offset must take account of the prefetch operation, which causes the PC to be 1 word (4 bytes) ahead of the current instruction.

Table 3-25. Summary of Branch Instruction

THUMB assembler	ARM equivalent	Action
B label	BAL label (halfword offset)	Branch PC relative +/- Offset11 << 1, where label is PC +/- 2048 bytes.

NOTE: The address specified by label is a full 12-bit two's complement address, but must always be halfword aligned (ie bit 0 set to 0), since the assembler places label >> 1 in the Offset11 field.

EXAMPLES

```
here    B here          ;  Branch onto itself. Assembles to 0xE7FE.
                        ;  (Note effect of PC offset).
        B jimmy         ;  Branch to 'jimmy'.
        •               ;  Note that the THUMB opcode will contain the number of
        •
        •               ;  halfwords to offset.
jimmy   •               ;  Must be halfword aligned.
```

FORMAT 19: LONG BRANCH WITH LINK

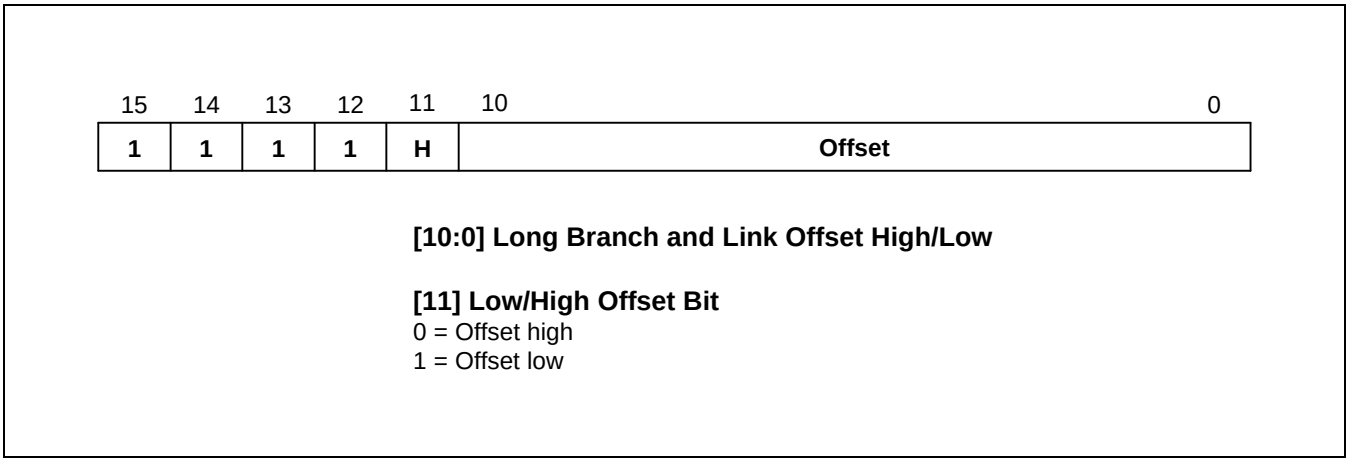


Figure 3-48. Format 19

OPERATION

This format specifies a long branch with link.

The assembler splits the 23-bit two's complement half-word offset specifed by the label into two 11-bit halves, ignoring bit 0 (which must be 0), and creates two THUMB instructions.

Instruction 1 (H = 0)

In the first instruction the Offset field contains the upper 11 bits of the target address. This is shifted left by 12 bits and added to the current PC address. The resulting address is placed in LR.

Instruction 2 (H =1)

In the second instruction the Offset field contains an 11-bit representation lower half of the target address. This is shifted left by 1 bit and added to LR. LR, which now contains the full 23-bit address, is placed in PC, the address of the instruction following the BL is placed in LR and bit 0 of LR is set.

The branch offset must take account of the prefetch operation, which causes the PC to be 1 word (4 bytes) ahead of the current instruction

INSTRUCTION CYCLE TIMES

This instruction format does not have an equivalent ARM instruction.

Table 3-26. The BL Instruction

L	THUMB assembler	ARM equivalent	Action
0	BL label	none	LR := PC + OffsetHigh << 12
1			temp := next instruction address PC := LR + OffsetLow << 1 LR := temp | 1

EXAMPLES

```
        BL faraway      ; Unconditionally Branch to 'faraway'
next    •               ; and place following instruction
        •               ; address, ie 'next', in R14,the Link
                        ; register and set bit 0 of LR high.
                        ; Note that the THUMB opcodes will
                        ; contain the number of halfwords to offset.
faraway •               ; Must be Half-word aligned.
        •
```

INSTRUCTION SET EXAMPLES

The following examples show ways in which the THUMB instructions may be used to generate small and efficient code. Each example also shows the ARM equivalent so these may be compared.

MULTIPLICATION BY A CONSTANT USING SHIFTS AND ADDS

The following shows code to multiply by various constants using 1, 2 or 3 Thumb instructions alongside the ARM equivalents. For other constants it is generally better to use the built-in MUL instruction rather than using a sequence of 4 or more instructions.

```
            Thumb       ARM
```

1. Multiplication by 2^n (1,2,4,8,...)

```
            LSL         Ra, Rb, LSL #n          ;   MOV Ra, Rb, LSL #n
```

2. Multiplication by 2^n+1 (3,5,9,17,...)

```
            LSL         Rt, Rb, #n              ;   ADD Ra, Rb, Rb, LSL #n
            ADD         Ra, Rt, Rb
```

3. Multiplication by 2^n-1 (3,7,15,...)

```
            LSL         Rt, Rb, #n              ;   RSB Ra, Rb, Rb, LSL #n
            SUB         Ra, Rt, Rb
```

4. Multiplication by -2^n (-2, -4, -8, ...)

```
            LSL         Ra, Rb, #n              ;   MOV Ra, Rb, LSL #n
            MVN         Ra, Ra                  ;   RSB Ra, Ra, #0
```

5. Multiplication by -2^n-1 (-3, -7, -15, ...)

```
            LSL         Rt, Rb, #n              ;   SUB Ra, Rb, Rb, LSL #n
            SUB         Ra, Rb, Rt
```

Multiplication by any C = {2^n+1, 2^n-1, -2^n or -2^n-1} * 2^n
Effectively this is any of the multiplications in 2 to 5 followed by a final shift. This allows the following additional constants to be multiplied. 6, 10, 12, 14, 18, 20, 24, 28, 30, 34, 36, 40, 48, 56, 60, 62 .....

```
            (2..5)                              ;   (2..5)
            LSL         Ra, Ra, #n              ;   MOV Ra, Ra, LSL #n
```

GENERAL PURPOSE SIGNED DIVIDE

This example shows a general purpose signed divide and remainder routine in both Thumb and ARM code.

Thumb code

```
;signed_divide                          ;  Signed divide of R1 by R0: returns quotient in R0,
                                        ;  remainder in R1

;Get abs value of R0 into R3
                ASR     R2, R0, #31     ;  Get 0 or -1 in R2 depending on sign of R0
                EOR     R0, R2          ;  EOR with -1 (0×FFFFFFFF) if negative
                SUB     R3, R0, R2      ;  and ADD 1 (SUB -1) to get abs value

;SUB always sets flag so go & report division by 0 if necessary
                BEQ     divide_by_zero

;Get abs value of R1 by xoring with 0xFFFFFFFF and adding 1 if negative
                ASR     R0, R1, #31     ;  Get 0 or -1 in R3 depending on sign of R1
                EOR     R1, R0          ;  EOR with -1 (0×FFFFFFFF) if negative
                SUB     R1, R0          ;  and ADD 1 (SUB -1) to get abs value

;Save signs (0 or -1 in R0 & R2) for later use in determining ; sign of quotient & remainder.
                PUSH     {R0, R2}

;Justification, shift 1 bit at a time until divisor (R0 value) ; is just <= than dividend (R1 value). To do this shift
dividend ; right by 1 and stop as soon as shifted value becomes >.
                LSR     R0, R1, #1
                MOV     R2, R3
                B       %FT0
just_l          LSL     R2, #1
0               CMP     R2, R0
                BLS     just_l
                MOV     R0, #0          ;  Set accumulator to 0
                B       %FT0            ;  Branch into division loop

div_l           LSR     R2, #1
0               CMP     R1, R2          ;  Test subtract
                BCC     %FT0
                SUB     R1, R2          ;  If successful do a real subtract
0               ADC     R0, R0          ;  Shift result and add 1 if subtract succeeded

                CMP     R2, R3          ;  Terminate when R2 == R3 (ie we have just
                BNE     div_l           ;  tested subtracting the 'ones' value).
```

```
Now fixup the signs of the quotient (R0) and remainder (R1)
                POP     {R2, R3}                ;  Get dividend/divisor signs back
                EOR     R3, R2                  ;  Result sign
                EOR     R0, R3                  ;  Negate if result sign = – 1
                SUB     R0, R3
                EOR     R1, R2                  ;  Negate remainder if dividend sign = – 1
                SUB     R1, R2
                MOV     pc, lr
```

ARM Code

```
signed_divide                                   ;  Effectively zero a4 as top bit will be shifted out later
                ANDS    a4, a1, #&80000000
                RSBMI   a1, a1, #0
                EORS    ip, a4, a2, ASR #32
;ip bit 31 = sign of result
;ip bit 30 = sign of a2
                RSBCS    a2, a2, #0

;Central part is identical code to udiv (without MOV a4, #0 which comes for free as part of signed entry sequence)
                MOVS    a3, a1
                BEQ     divide_by_zero

just_l                                          ;  Justification stage shifts 1 bit at a time
                CMP     a3, a2, LSR #1
                MOVLS   a3, a3, LSL #1          ;  NB: LSL #1 is always OK if LS succeeds
                BLO     s_loop

div_l
                CMP     a2, a3
                ADC     a4, a4, a4
                SUBCS   a2, a2, a3
                TEQ     a3, a1
                MOVNE   a3, a3, LSR #1
                BNE     s_loop2
                MOV     a1, a4
                MOVS    ip, ip, ASL #1
                RSBCS   a1, a1, #0
                RSBMI   a2, a2, #0
                MOV     pc, lr
```

DIVISION BY A CONSTANT

Division by a constant can often be performed by a short fixed sequence of shifts, adds and subtracts.

Here is an example of a divide by 10 routine based on the algorithm in the ARM Cookbook in both Thumb and ARM code.

Thumb Code

```
udiv10                                  ;  Take argument in a1 returns quotient in a1,
                                        ;  remainder in a2
        MOV     a2, a1
        LSR     a3, a1, #2
        SUB     a1, a3
        LSR     a3, a1, #4
        ADD     a1, a3
        LSR     a3, a1, #8
        ADD     a1, a3
        LSR     a3, a1, #16
        ADD     a1, a3
        LSR     a1, #3
        ASL     a3, a1, #2
        ADD     a3, a1
        ASL     a3, #1
        SUB     a2, a3
        CMP     a2, #10
        BLT     %FT0
        ADD      a1, #1
        SUB     a2, #10
0
        MOV      pc, lr
```

ARM Code

```
udiv10                                  ;  Take argument in a1 returns quotient in a1,
                                        ;  remainder in a2
        SUB     a2, a1, #10
        SUB     a1, a1, a1, lsr #2
        ADD     a1, a1, a1, lsr #4
        ADD     a1, a1, a1, lsr #8
        ADD     a1, a1, a1, lsr #16
        MOV     a1, a1, lsr #3
        ADD     a3, a1, a1, asl #2
        SUBS    a2, a2, a3, asl #1
        ADDPL   a1, a1, #1
        ADDMI   a2, a2, #10
        MOV     pc, lr
```

4 SYSTEM MANAGER

OVERVIEW

The KS17C4000 System Manager has the following functions:

— Arbitrates bus access requests from several master blocks, based on a fixed priority.

— Provides the memory control signals required for external memory accesses. For example, if a master block such as DMA or the CPU generates an address that corresponds to a DRAM bank, the System Manager's DRAM controller generates the required DRAM access signals (nRAS, nCAS, and so on).

— Supports big-endian mode. The internal system and the external memory are fixed in big-endian mode.

— Compensates for differences in bus width for data flowing between the external data bus and the internal data bus.

SYSTEM MANAGER REGISTERS

The KS17C4000 microcontroller has the SFRs, Special Function Registers, that keep the system control information of system manager, cache, DMA, UART and so on. The SFRs have the SMR, System Manager Register files, that configure the external memory maps such as DRAM, SRAM, ROM and extra-I/O control.

By utilizing the SMR, you can specify the memory type, external bus width, access cycles, required control signal timings (nRAS, nCAS and so on), memory bank location and each memory bank size which has configurable address space. The SMR provides (or accepts) the control signals, addresses, and data required by external devices during normal system operation. There are seven registers that control memory banks (ROM, SRAM, DRAM), extra-device, and DRAM Refresh.

The KS17C4000 provides up to 32 Mbytes of address space and each bank provides up to 16 Mbytes of memory space. Each bank can have 24 address pins and 8/16-bit data width.

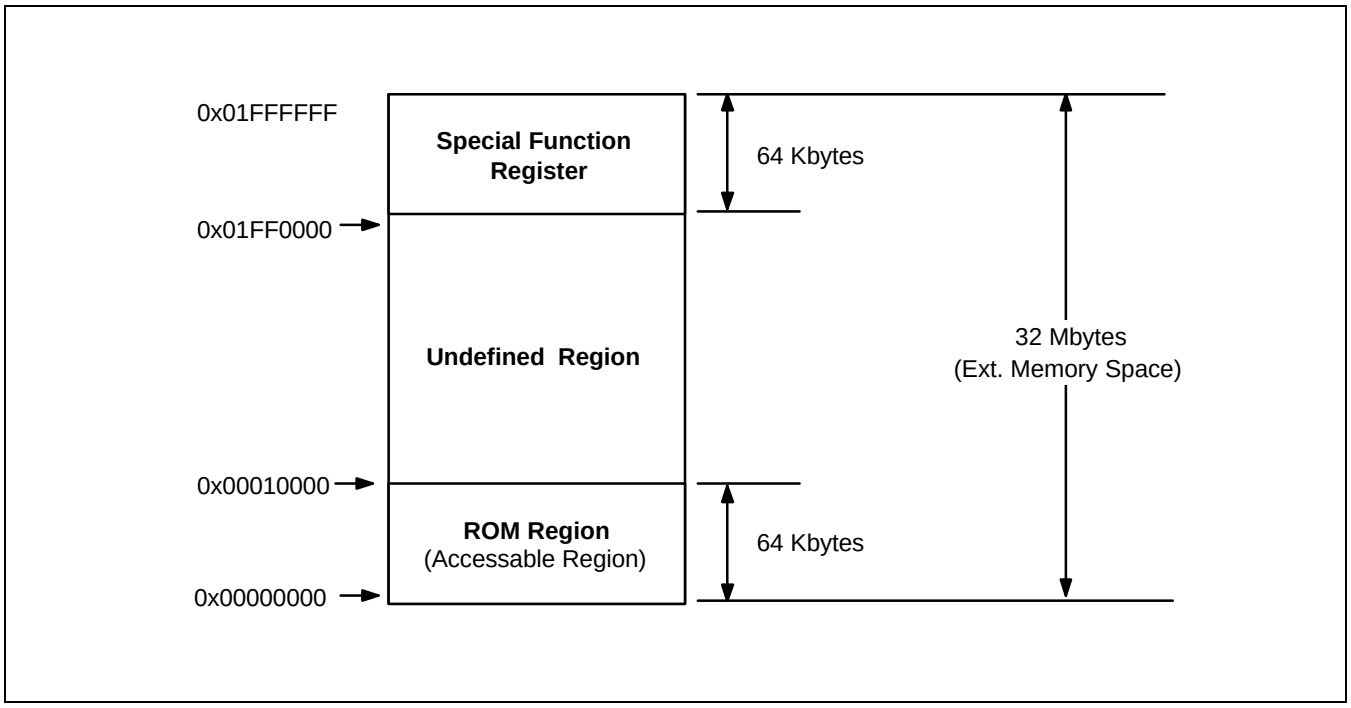


Figure 4-1. KS17C4000 Memory Map (Default Map after Reset)

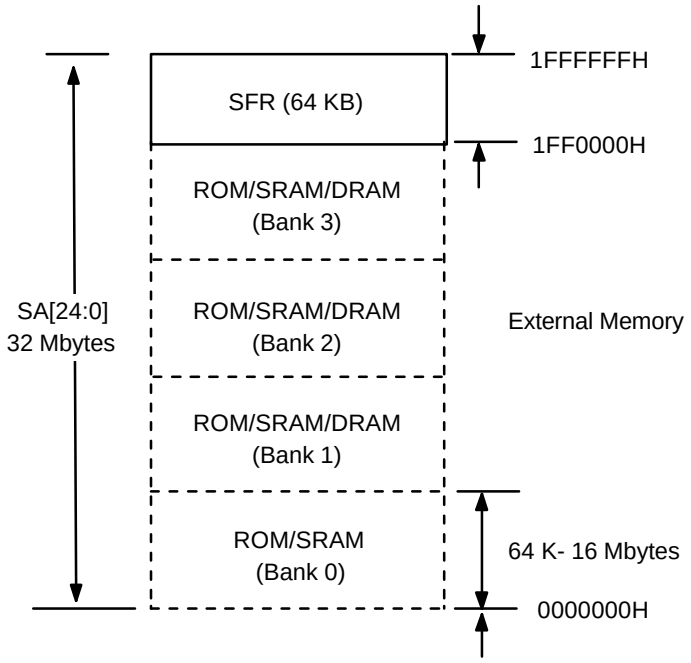
The KS17C4000 provides 32-Mbyte memory space and 25-bit internal system address bus. You can use any address area from 000 0000h to 1FF FFFFh in 64-Kbyte address steps. Each bank can be located anywhere in the 32-Mbyte address space, except the upper 64-Kbyte area where the SFRs (Special Function Registers) are located. To use the full 32-Mbyte memory space, we recommend that you allocate the SFRs to the upper 64-kbyte address area, 1FF0000h-1FFFFFFh.

The configurable memory allocation in the KS17C4000 is designed to provide you with convenience. By manipulating the SMRs, you can easily allocate the memory area anywhere you want and use the consecutively connected memory space without changing the H/W. You are also allowed to easily change the physical DRAM memory size by manipulating the SMR.

For example, if you want to change the size of memory space from 1 M half-word to 4 M half-word, you can enlarge the memory space just by changing the next pointer of the DRAM bank.

NOTE

The last 64 Kbyte area can not be allocated for memory banks, other than the SFRs. Because the last 64 Kbyte bank starts its address with 1FFxxxxh, the next pointer of the last bank should be 200 xxxxh. Actually, the next point is 9 bits, so the value of the next pointer is 000 xxxxh. If you need to utilize the full 32 M bytes of memory space, it is recommended that you allocate the SFRs to the last 64-Kbyte area, 1FF 0000h–1FF FFFFh, and use the remaining areas for other banks.



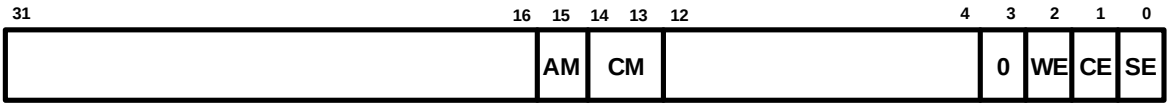
NOTE: When a DRAM is selected, it can be accessed to 4 Mbytes space of DRAM.

Figure 4-2. System Memory Map

SYSTEM REGISTER ADDRESS CONFIGURATION REGISTER (SYSCFG)

The SMRs (System Manager Registers) have the SYSCFG (System Register Address Configuration Register), which determines the start address (base point) of SFR (Special Function Register) files. The SYSCFG contains the start address of the SFRs. If the reset value of SYSCFG is 3FF1h, the SYSCFG is mapped to the virtual address 01FF 1000h.

Register	Offset Address	R/W	Description	Reset Value
SYSCFG	0x1000	R/W	Special function register to determine the start address	0x3FF1



[0] Stall Enable (ST)
When this bit is set to 0, stall operation is enabled.
0 = Disable; It is recommended for faster operation.
1 = Enable; Insert an internal wait inside the core logic when non-sequential memory accesses occur.

[1] Cache Enable (CE)
When this bit is set to 1, cache operation is enabled.
0 = Disable the cache operation.
1 = Enable the cache operation.

[2] Write Buffer Enable (WE)
When this bit is set to 1, writing buffer operation is enabled.
0 = Disable the writing buffer operation.
1 = Enable the writing buffer operation.

[3] Always 0

[12:4] SYSCFG Address (SFRs Start Address)
The resolution is 64 Kbytes, if you want to place the start address at 1800000h.
Setting value = 1800000h/10000h = 180h

[14:13] Cache Mode Selection Bits
00 = 2 KB SRAM, 2 KB cache
01 = 0 KB SRAM, 4 KB cache
10 = 4 KB SRAM, 0 KB cache

[15] Address Mux Enable Bit
0 = Normal mode (A8-A15)
1 = Address muxing mode (A16-A23/A8-A15)

Figure 4-3. System Register Address Configuration Register (SYSCFG)

Start Address

The SYSCFG[12:4] bits indicate the start address of the SFRs.

You can allocate the SFRs to the arbitrary location by manipulating the SYSCFG. We recommend that you do not change the SYSCFG during operation once you have configured it after a system reset. The SYSCFG should not be overlapped with any other bank.

If the start address of the SYSCFG is changed, other control registers in the SFRs will have a new address, which is the sum of its offset address and the new address of the SYSCFG. For example, after a system reset, the initial address of the SYSCFG is 1FF 1000h and the ROM control register has the initial address, '1FF 2000h', because the ROM control register has the offset value '2000h', and the initial address is the sum of 1FF 0000h and 2000h. If the SYSCFG is changed to 180h, the address of the ROM control register will be 180 2000h.

Cache Disable/Enable

To enable or disable the cache, you should set the cache enable (CE) bit of the SYSCFG register to "1" or "0", respectively. Because cache memory does not have an auto-flush feature, you must be careful to verify the coherency of data whenever you re-enable the cache. You must also carefully check any change that the DMA controller may make to data stored in memory. (Usually, the memory area allocated to DMA access operations must be non-cacheable.)

The internal 4-Kbyte SRAM can be used as a cache area. To configure this area, you should use the cache mode bits, SYSCFG[14:13]. If you do not need to use the entire 4-Kbyte area as cache, you can use the remaining area as the internal SRAM. This area is accessed using the address of the SFR in the internal SRAM field.

Write Buffer Disable/Enable

The KS17C4000 has four programmable write buffer registers that are used to improve the speed of memory write operations. When you enable a write buffer, the CPU writes data into the write buffer, instead of an external memory location. This saves the cycle that would normally be required to complete the external memory write operation. The four write buffers also enhance the performance of the ARM7TDMI core store operations.

To maintain data coherency between the cache and external memory, the KS17C4000 uses a write-through policy. An internal 4-level write buffer compensates for performance degradation caused by write-throughs. (For more information, read Chapter 12.)

EXTERNAL MEMORY CONTROL SPECIAL REGISTERS

MEMORY CONTROL REGISTER 0, 1, 2, 3

Register	Offset Address	R/W	Description	Reset Value
MEMCON0	0x2000	R/W	Memory control register 0 (for ROM/SRAM)	0080 0600h
MEMCON1	0x2004	R/W	Memory control register 1 (for ROM/SRAM/DRAM)	0000 0000h
MEMCON2	0x2008	R/W	Memory control register 2 (for ROM/SRAM/DRAM)	0000 0000h
MEMCON3	0x200c	R/W	Memory control register 3 (for ROM/SRAM/DRAM)	0000 0000h

[0]	Memory Selection (MS)	0 = ROM/SRAM bank 1 = DRAM bank (MEMCON0, always 0)
[1]	Bus Width (DW)	0 = 8 bits (byte) 1 = 16 bits (half-word) This bit is read only when MEMCON0 is accessed.
[3:2]	Page mode or Col Address	When ROM/SRAM memory, (Page Mode Configuration-Pmc) 00 = Normal 01 = 4 data page 10 = 8 data page 11 = 16 data page When DRAM is configured, (Column Address Number-CAN) 00 = 8 bits 01 = 9 bits 10 = 10 bits 11 = 11 bits
[4]	EDO DRAM / x16 SRAM	When DRAM is configured, EDO DRAM (Not built-in MEMCntr0) 0 = Ordinary 1 = EDO DRAM. When SRAM is configured, SRAM (Not built-in MEMCntr0) 0 = Ordinary (or ROM) 1 = x16 type SRAM.
[6:5]	Access cycle, CAS time	When ROM memory, (Page Mode Access Cycles-Tacp) 00 = 5 cycles 01 = 2 cycles 10 = 3cycles 11 = 4 cycles When DRAM is configured, CAS Strobe time (@ Page mode) 00 = 1 cycle 01 = 2 cycles 10 = 3 cycles 11 = 4 cycles
[7]	CAS pre-charge(Tcp)	When DRAM is configured, CAS Pre charge time (Tcp) 0 = 1 cycle 1 = 2 cycles.

[10:8]	Access cycles of Bank	When ROM memory is selected, (Tacc) 000 = Disable bank 001 = 2 cycles 010 = 3 cycles 011 = 4 cycles 100 = 5 cycles 101 = 6 cycles 110 = 7 cycles 111 = Not used When SRAM memory is selected, (Tacc) 000 = Disable bank 001 = 2 cycles 010 = 3 cycles 011 = 4 cycles 100 = 5 cycles 101 = 6 cycles 110 = 7 cycles 111 = Not used When DRAM memory is selected, access time in CAS strobe mode (@ Single mode - Tcs) 000 = 1 cycles 001 = 2 cycles 010 = 3 cycles 011 = 4 cycles 100 = 5 cycles 101 = Not used 110 = Not used 111 = Disable Bank
[11]	RAS to CAS delay (Trc)	When DRAM is configured, RAS to CAS delay 0 = 1 cycle 1 = 2 cycles
[13:12]	RAS pre-charge time (Trp)	When DRAM memory is configured, 00 = 1 cycle 01 = 2 cycles 10 = 3 cycles 11 = 4 cycles
[22:14]	Base Address (BA)	Indicates the bank start address. You can configure the bank size in 64KB unit. If the bank start address is 0x01f0000, the base address of this bank is 0x1f.
[31:23]	End Address (EA)	Indicates the bank end address + 1. If the bank end address is 0x0ffffff, the end address is 0x100: 0x0ff+1.

Address Bus Generation

The address bus of the KS17C4000 is quite different from the general MCUs'. Although general MCU does not use the A0 pin in 16-bit data bus width, the KS17C4000 always uses the A0 pin regardless of data bus width. When an 8-bit data bus is selected, the resolution of address bus is a byte and when a 16-bit is selected, the resolution of address bus is a half-word.

Data bus width	External Address Pins (ADDR[23:0])	Accessible Memory size
8-bits	A23-A0 (internal)	16 M bytes
16-bits	A24-A1 (internal)	16 M half-word (32 Mbytes)

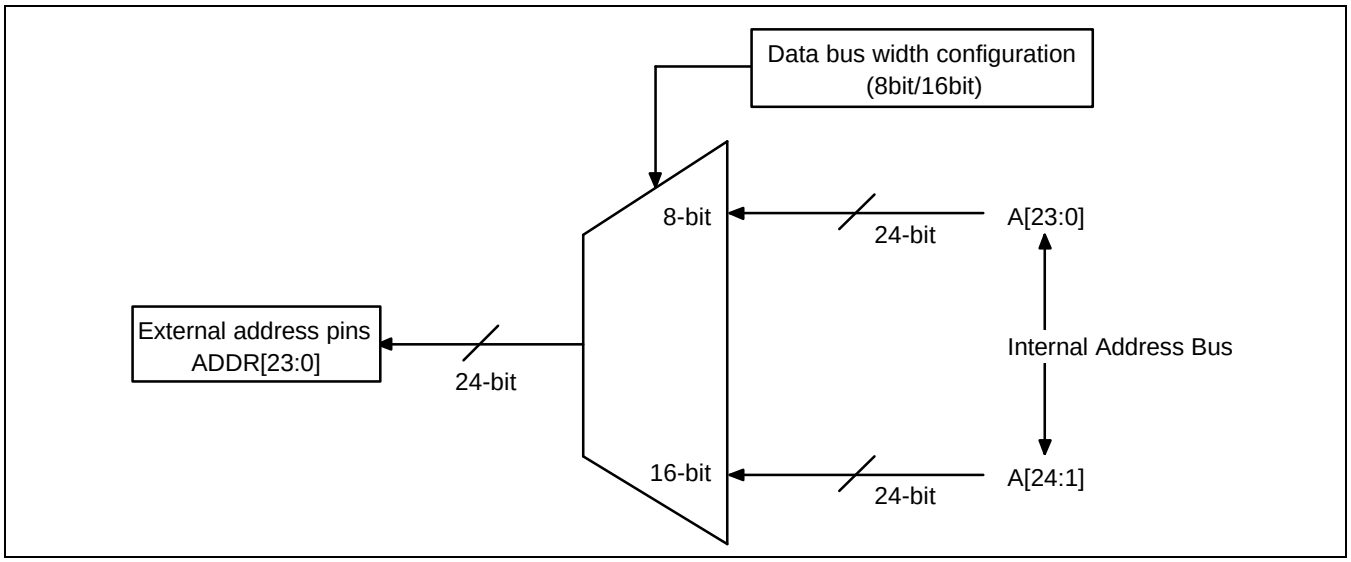


Figure 4-4. External Address Bus Generation (ADDR[23:0])

SRAM Bank Configuration

The nWBE (not Write Byte Enable) signal is for SRAM, the extra I/O or external memories such as flash memory. When the CPU writes one-byte data to an external RAM with 16-bit data bus, only one 8-bit data can be written. Other 8-bit data can not be written. nWBE[0] is for low-byte write operation and nWBE[1] is for upper-byte write operation.

The DRAM has writing methods different from SRAM or other external memories. The DRAM module has two CAS signals that separate data bus in a byte unit order. A RAS signal is used for bank selection and a CAS signal for byte selection.

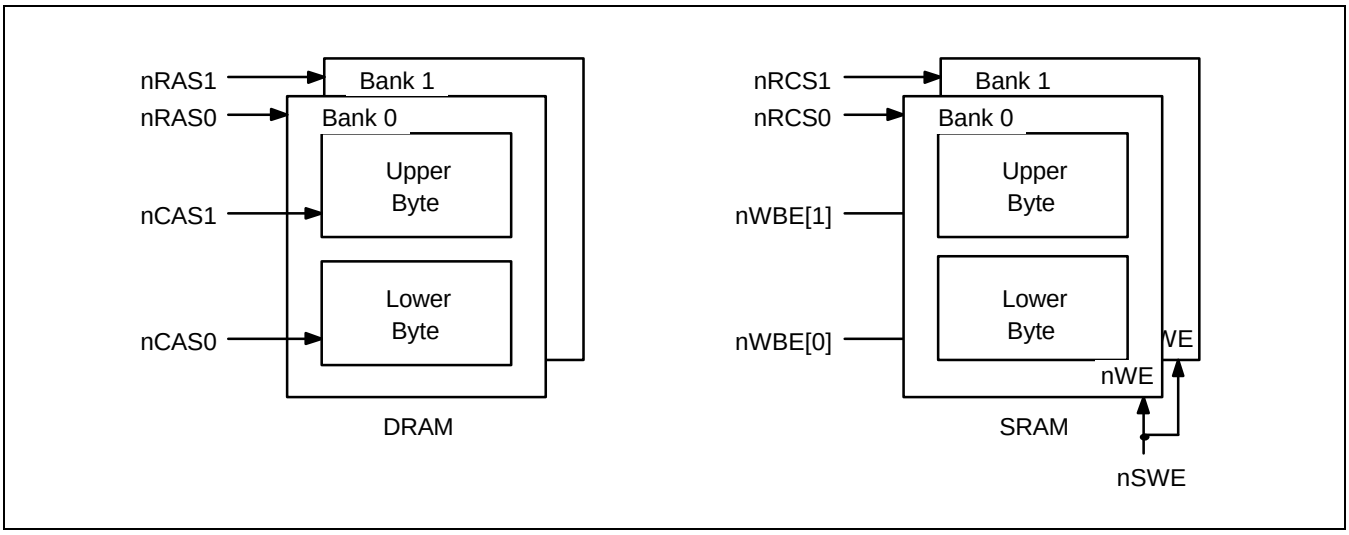


Figure 4-5. DRAM(x16) and SRAM(x16) Bank Configuration (For x16 data bus)

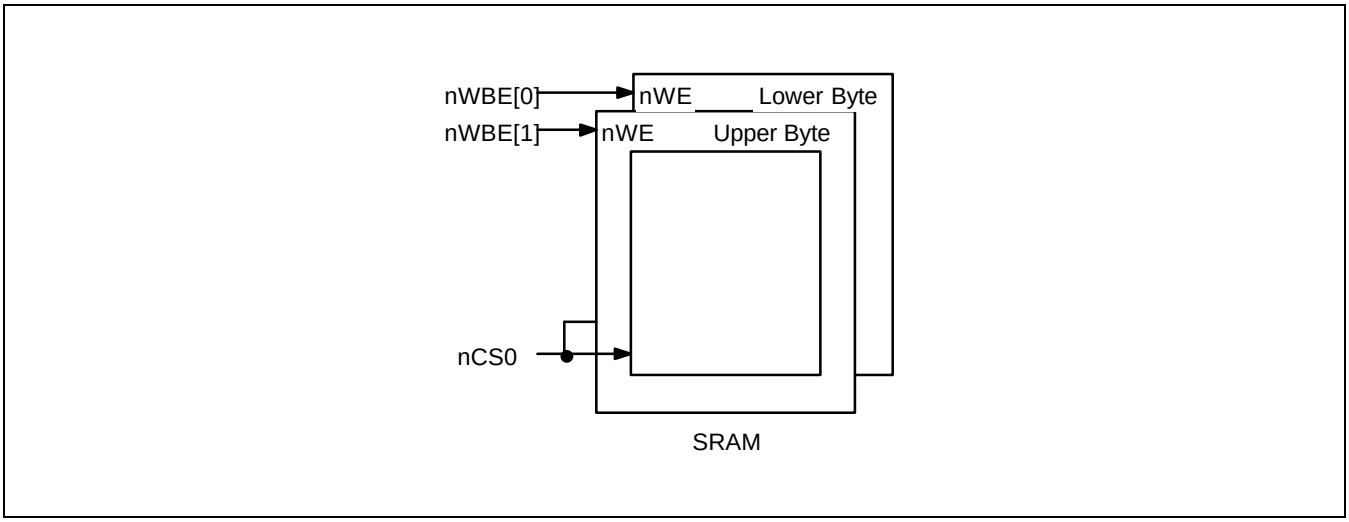


Figure 4-6. SRAM(×8) Configuration (For × 16 data bus)

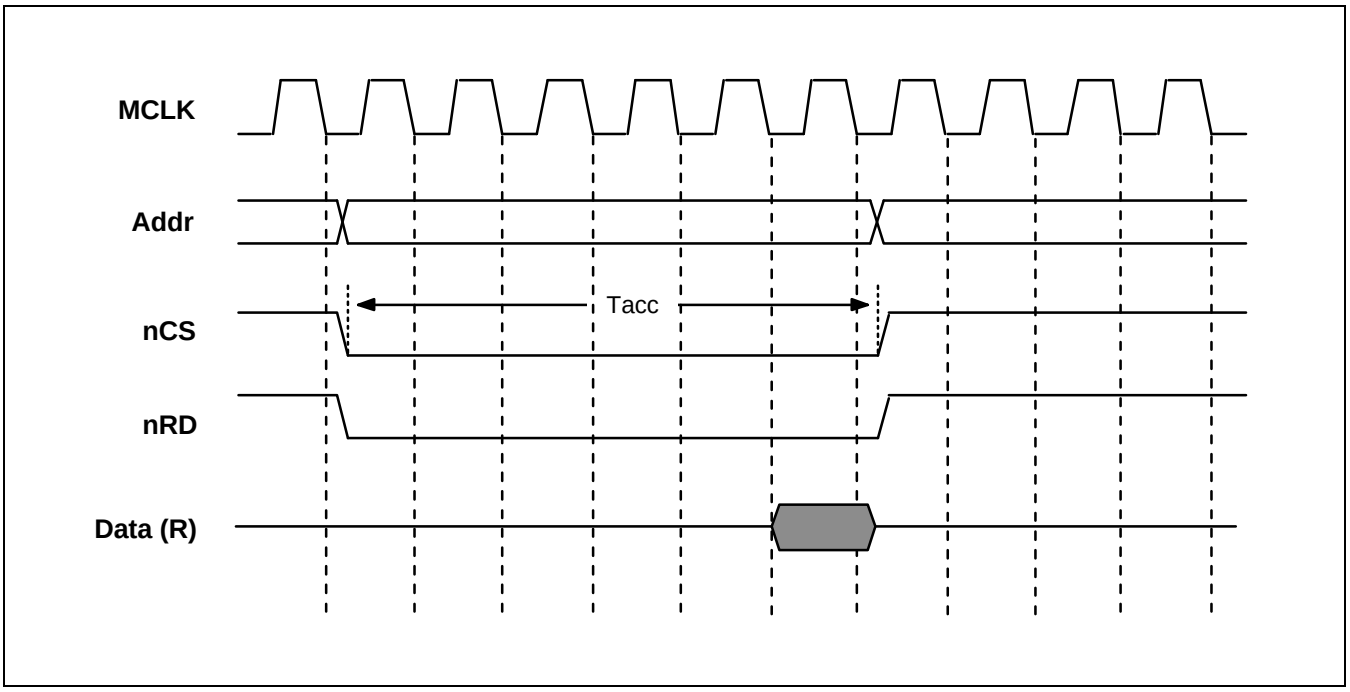


Figure 4-7. ROM/SRAM/Flash READ Access Timing

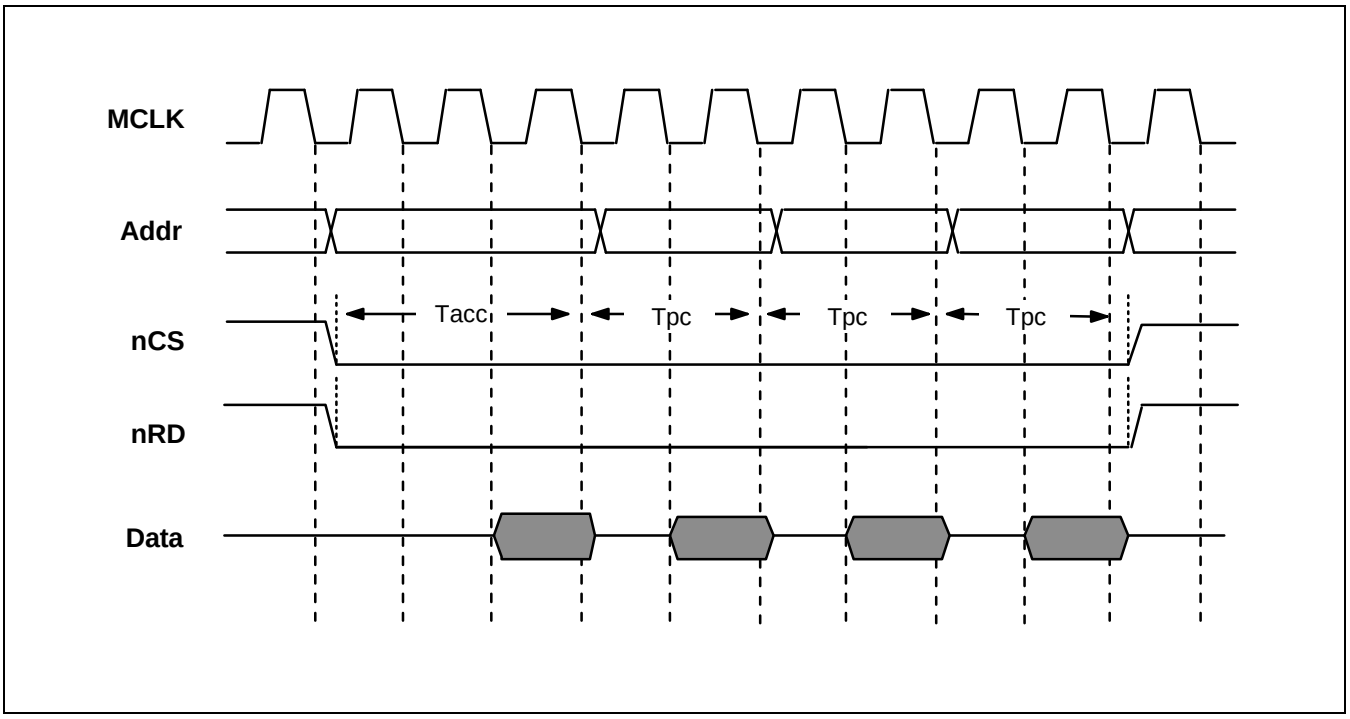


Figure 4-8. ROM/Flash Page READ Access Timing

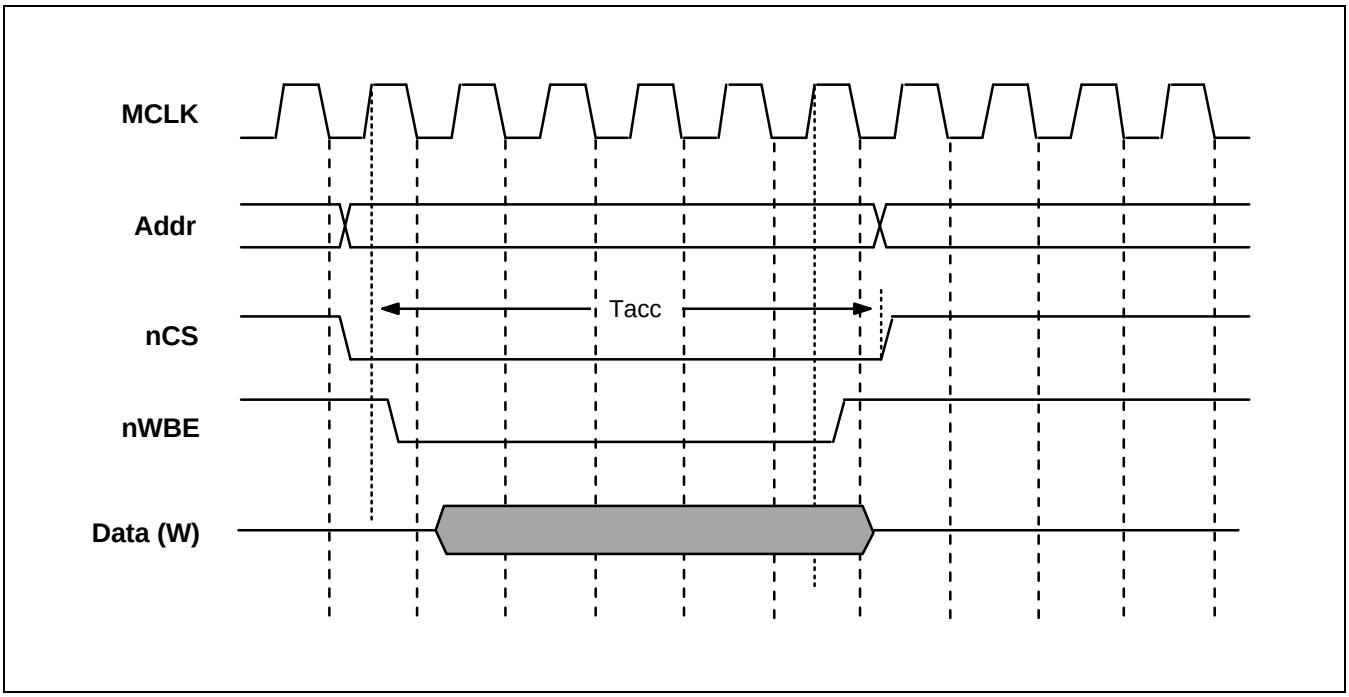


Figure 4-9. ROM/SRAM/Flash Write Access Timing

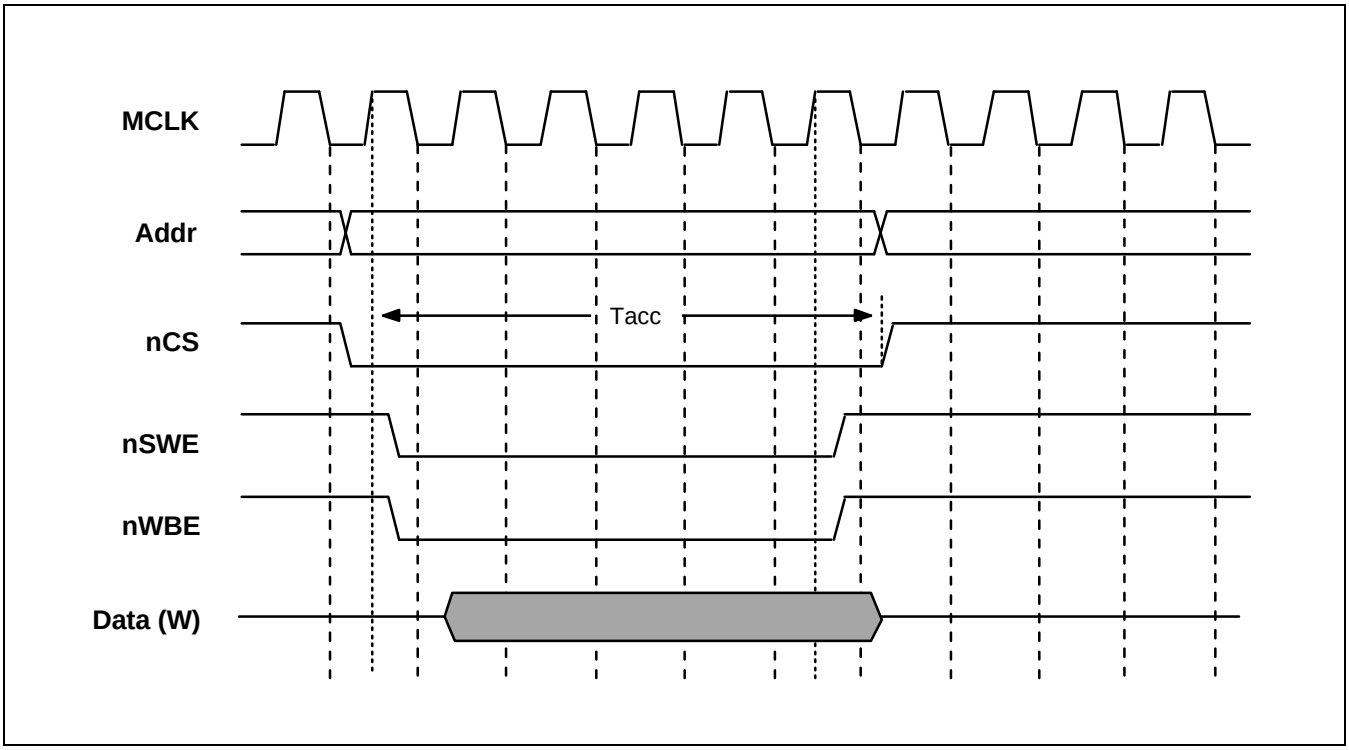


Figure 4-10. x16 SRAM Write Access Timing

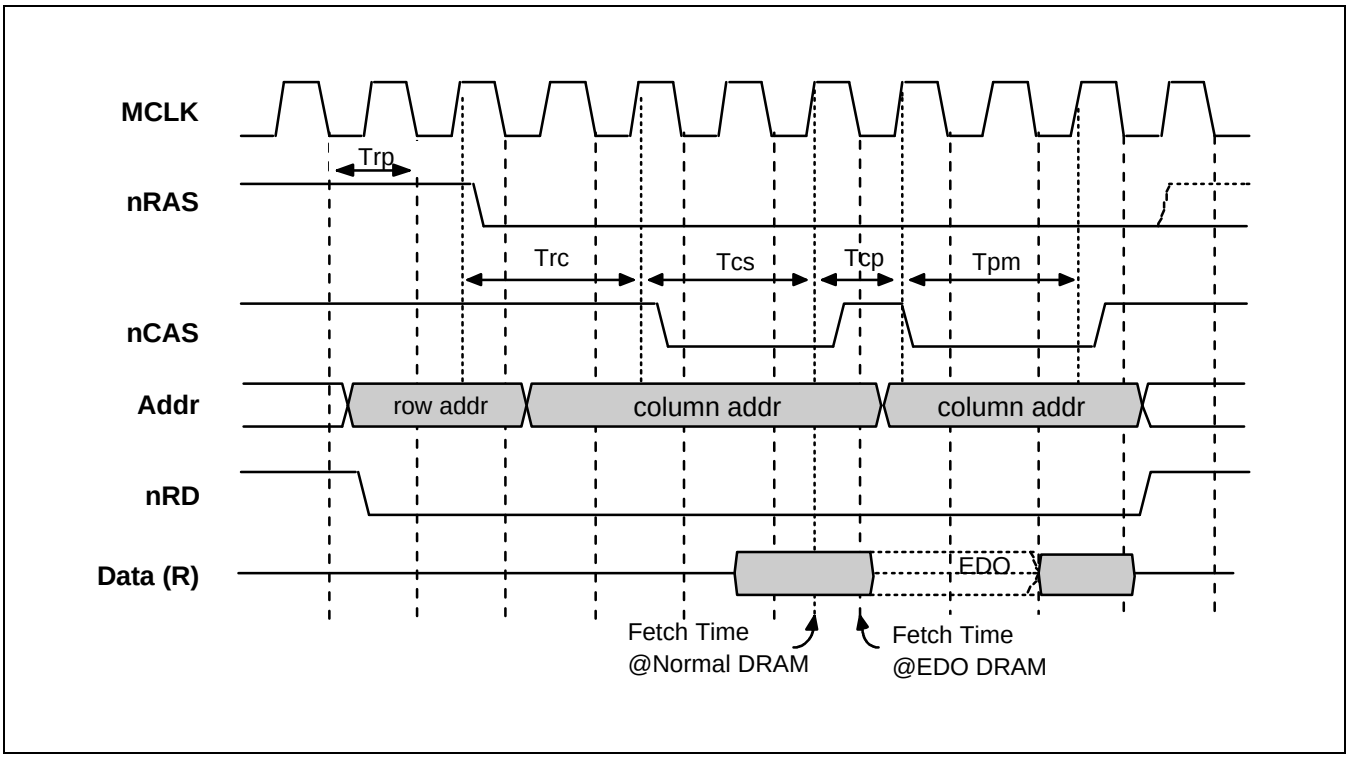


Figure 4-11. DRAM Bank Read Timing (Page Mode)

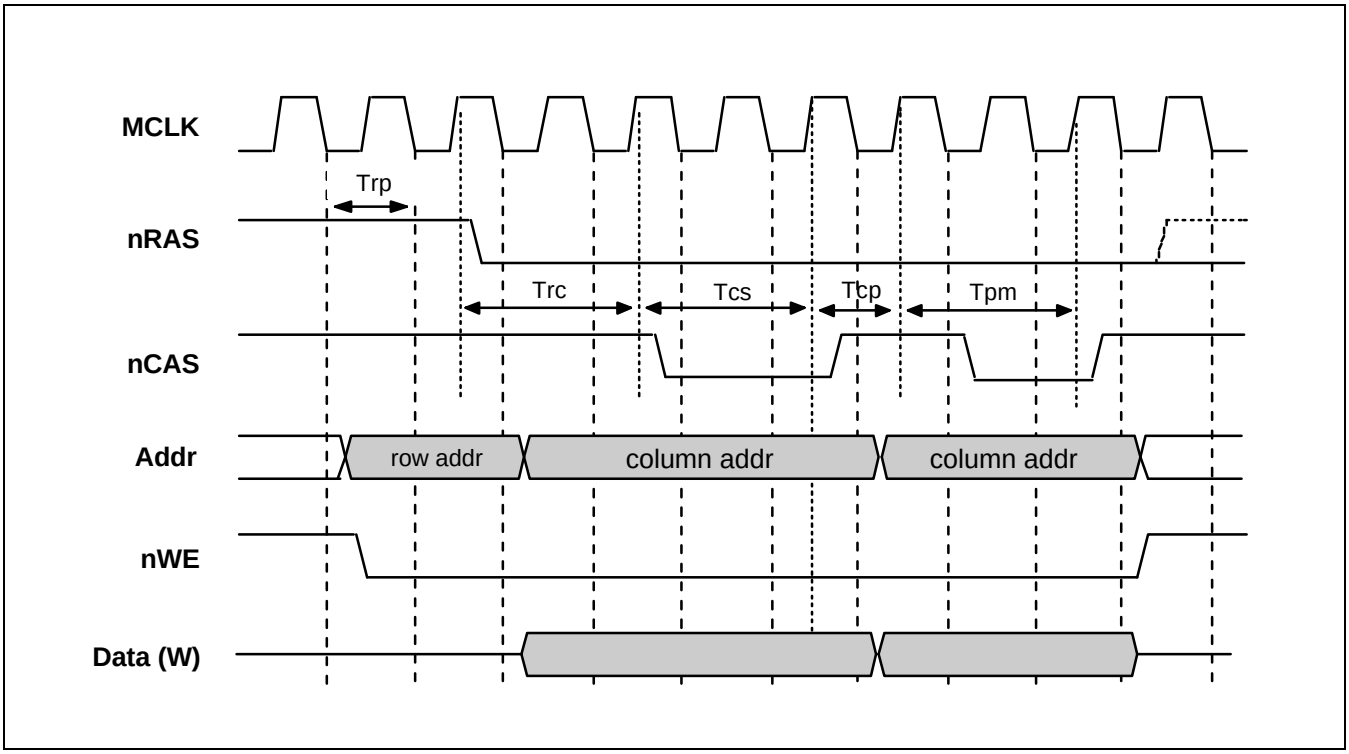


Figure 4-12. DRAM Bank Write Timing (Page Mode)

EXTRA DEVICE CONTROL REGISTER

The KS17C4000 supports an external device control without a glue logic. It is very cost-effective because no address decoding logic is needed. This method allows you to use an external output port or simple chip select function.

EXTRA DEVICE CONTROL REGISTER

The CPU can allocate address blocks not only to the four ROM/SRAM/DRAM banks, but also to extra device address. Using the System Manager registers, you can therefore configure special extra device addresses which provides the low-cost external device control solutions.

Memory accesses to the extra device address is controlled by the extra device control registers, EDVCON.

Register	Offset Address	R/W	Description	Reset Value
EDVCON	0x2014	R/W	Extra device control register	0000h

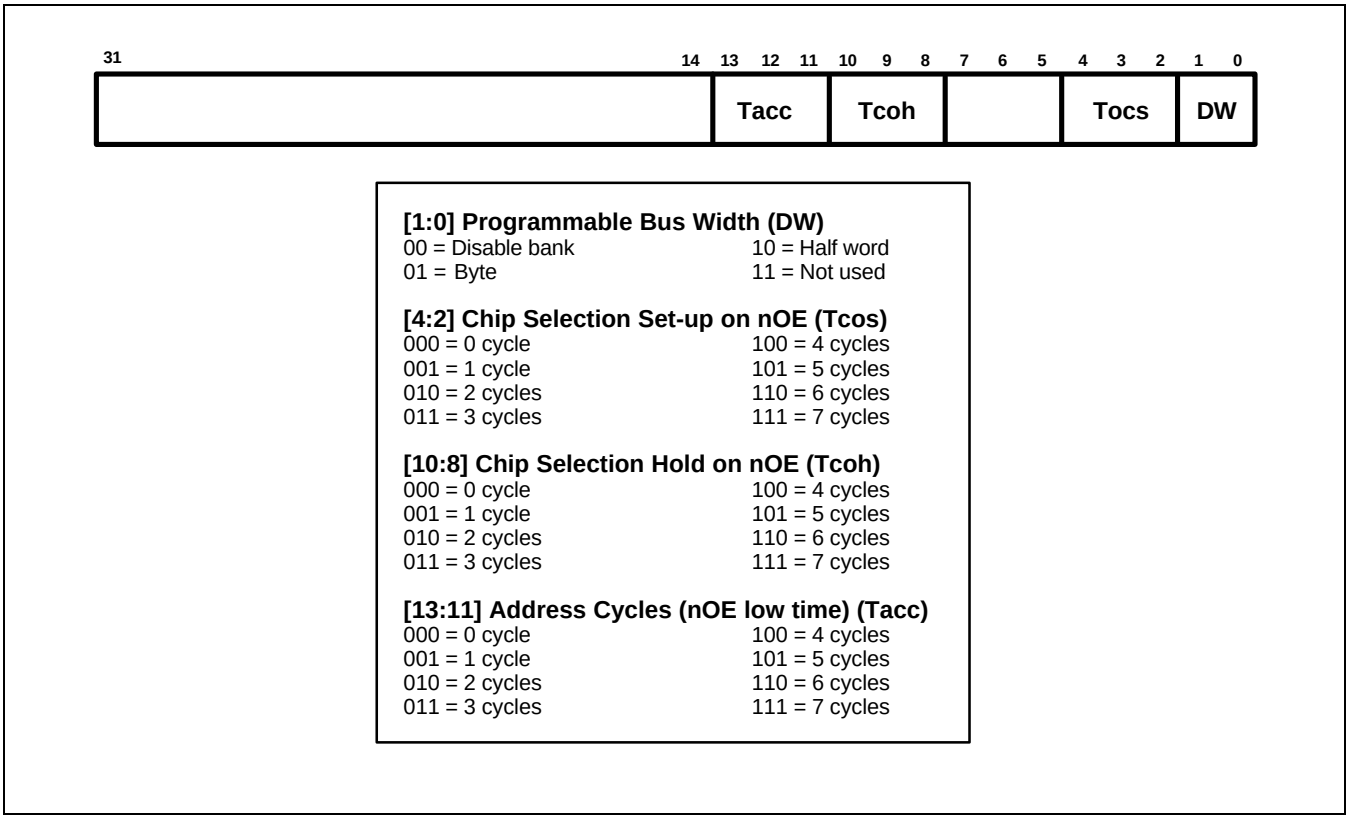


Figure 4-13. Extra Device Control Registers (EDVCON)

EXTERNAL OUTPUT PORT REGISTER

When you add an external output latch device as an output port, nWREXP signal don't require extra address decoding logic chips. The Memory accesses to the external Output address is made through EXTPORT register. Access time is controlled by the extra device control register, EDVCON. When MDS mode is selected, this pin is used as a write strobe signal of external output latch, which is emulated for port 0 output.

Register	Offset Address	R/W	Description	Reset Value
EXTPORT	0x201a	R/W	External Port data register	0000h

EXTRA CHIP SELECTION DATA REGISTER

When you use an external device as an external memory, ECS0 and ECS1 signal don't require extra address decoding logic chips. The Access to the external device is made through ECSDATA0 or ECSDATA1. If DRAM bank is selected and CAS0 or CAS1 is used, ECS0/ECS1 does not support extra chip selection function. Access time is controlled by the extra device control register, EDVCON.

Register	Offset Address	R/W	Description	Reset Value
ECSDATA0	0x201c	R/W	Extra chip selection data register	0000h
ECSDATA1	0x201e	R/W	Extra chip selection data register	0000h

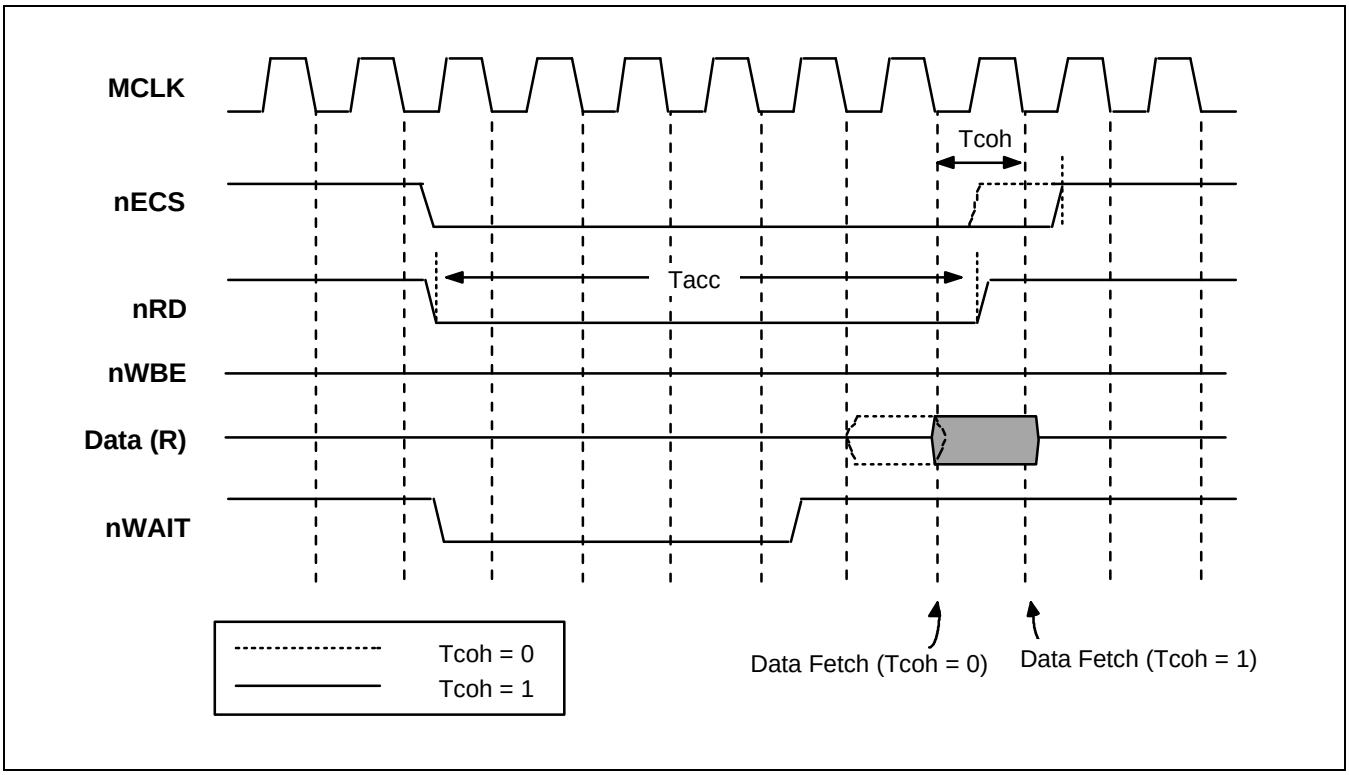


Figure 4-14. ECSDATA 0/1 Read Timing with nWAIT (Tcoh = 1, Tacc = 1, Tcos = 0, Tacs = 1)

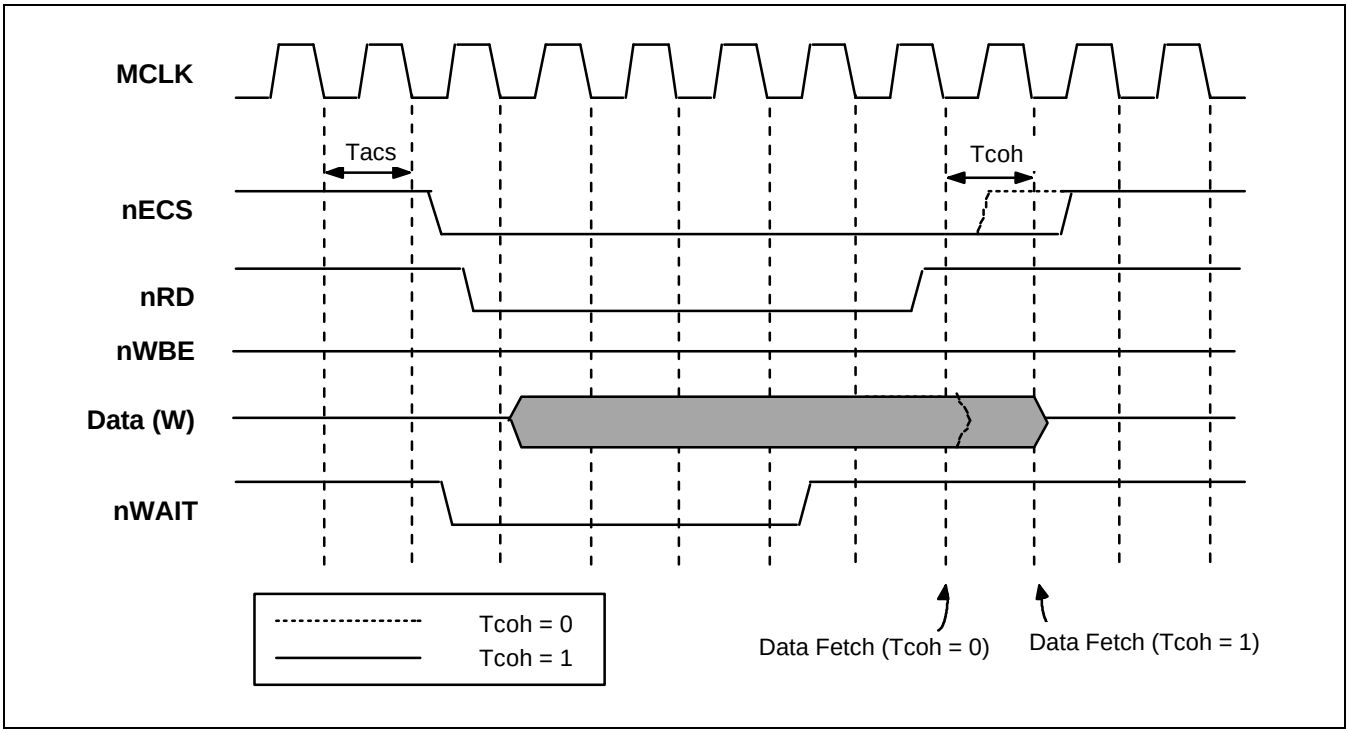


Figure 4-15. ECSDATA 0/1 Write Timing with nWAIT (Tcoh = 1, Tacc = 1, Tcos = 0, Tacs = 1)

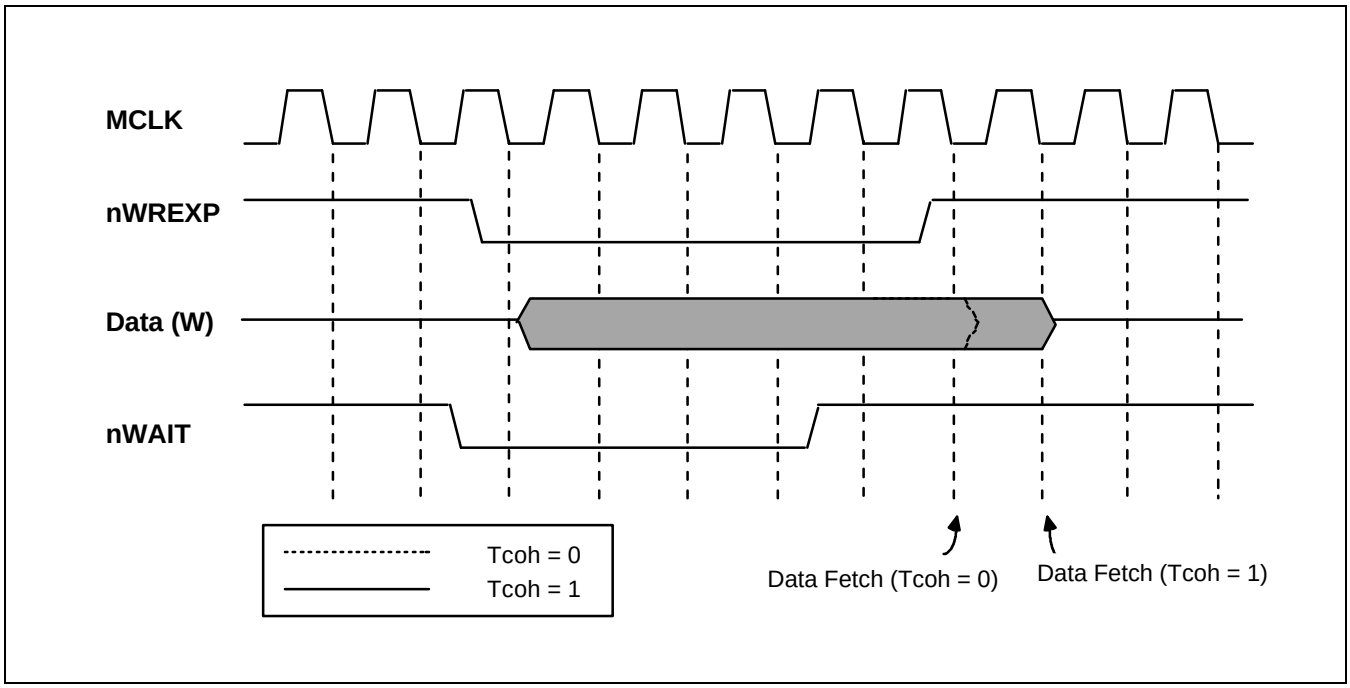
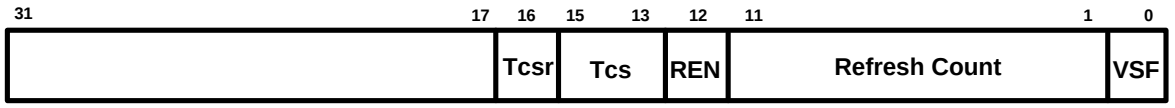


Figure 4-16. EXTPORT Write Timing with nWAIT (Tcoh = 1, Tacc = 1, Tcos = 0, Tacs = 1)

DRAM REFRESH CONTROL REGISTER

The KS17C4000 DRAM interface provides the CAS before entering RAS (CBR) refresh and Self refresh mode. The refresh control register (REFCON) determines refresh mode, refresh timings, refresh intervals as well as external bus enable. The KS17C4000, however, does not provide the hardware self refresh mode.

Register	Offset Address	R/W	Description	Reset Value
REFCON	0x2010	R/W	DRAM refresh control register	0 0001h



[0] Validity of Special Regiater Field (VSF)
0 = Not accessable to memory bank
1 = Accessable to memory bank
(Whenever MCU accesses one of systen manager registers (SMR), VSF bit is auto-cleared and external bus disabled. To re-activate, VSF bit should not set to 1 by using STMIA instruction in the last. Programmer should write into 5 system managers altogether with STMIA instruction while VSF bit set instruction is in the last.)

[11:1] Refresh interval (Refresh Count)
(Whenever a time-out occurs, this 11-bit value is automatically reloaded into the refresh counter. It defines theduration of the refresh cycle in MCLKs.)
Refresh Period = (2 ^{11} – value + 1)/MCLK

[12] Address Set-up before nECS (Tacs)
0 = Self refresh mode or disable DRAM refresh.
(When this bit is set to 0, DRAM enters the self refresh mode and can not be accessed.)
1 = Enable DRAM refresh.
(When this bit is set to 1, MCU refreshes DRAM periodically and can read/write the DRAM.)

[15:13] CAS Hold Time (Tcs)

000 = 1 cycle	001 = 2 cycle
010 = 3 cycles	011 = 4 cycles
100 = 5 cycles	Others = Not used

[16] CAS Set-up Time (Tcsr)
0 = 1 cycle
1 = 2 cycles

Figure 4-17. DRAM Refresh Control Register (REFCON)

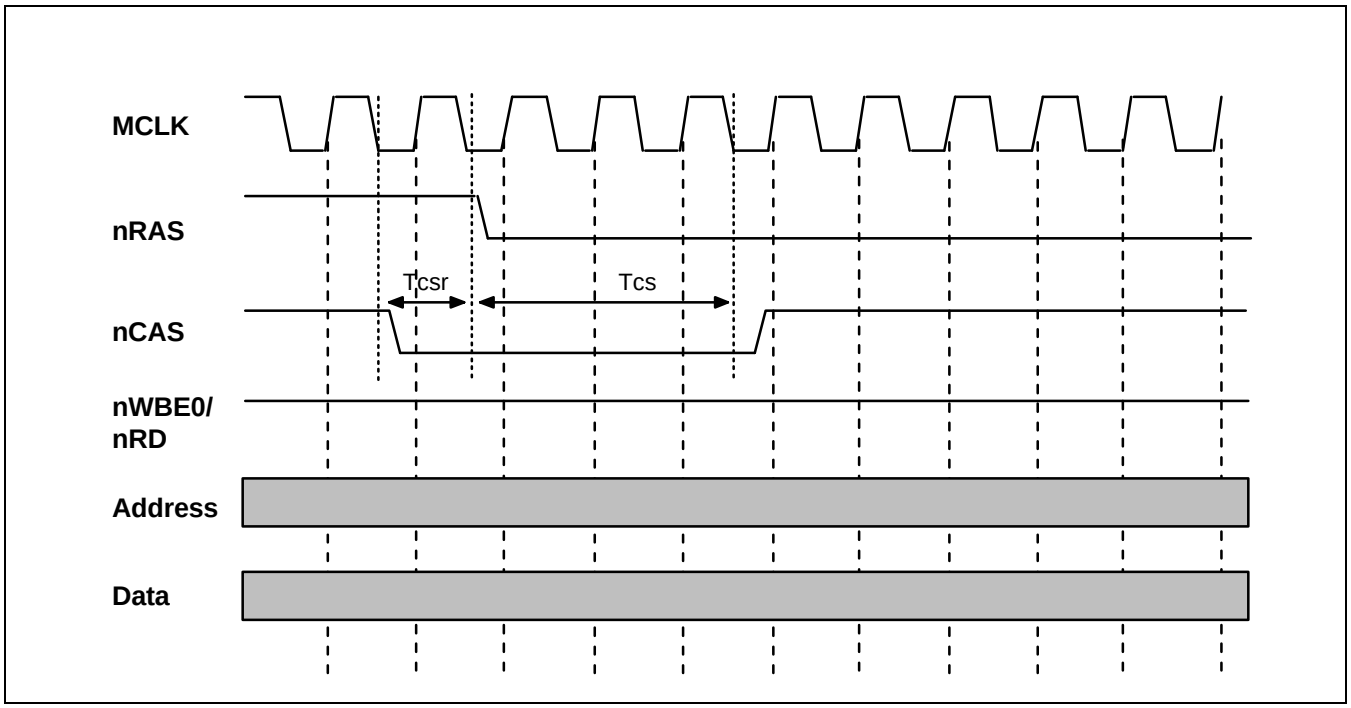


Figure 4-18. DRAM Refresh Timing

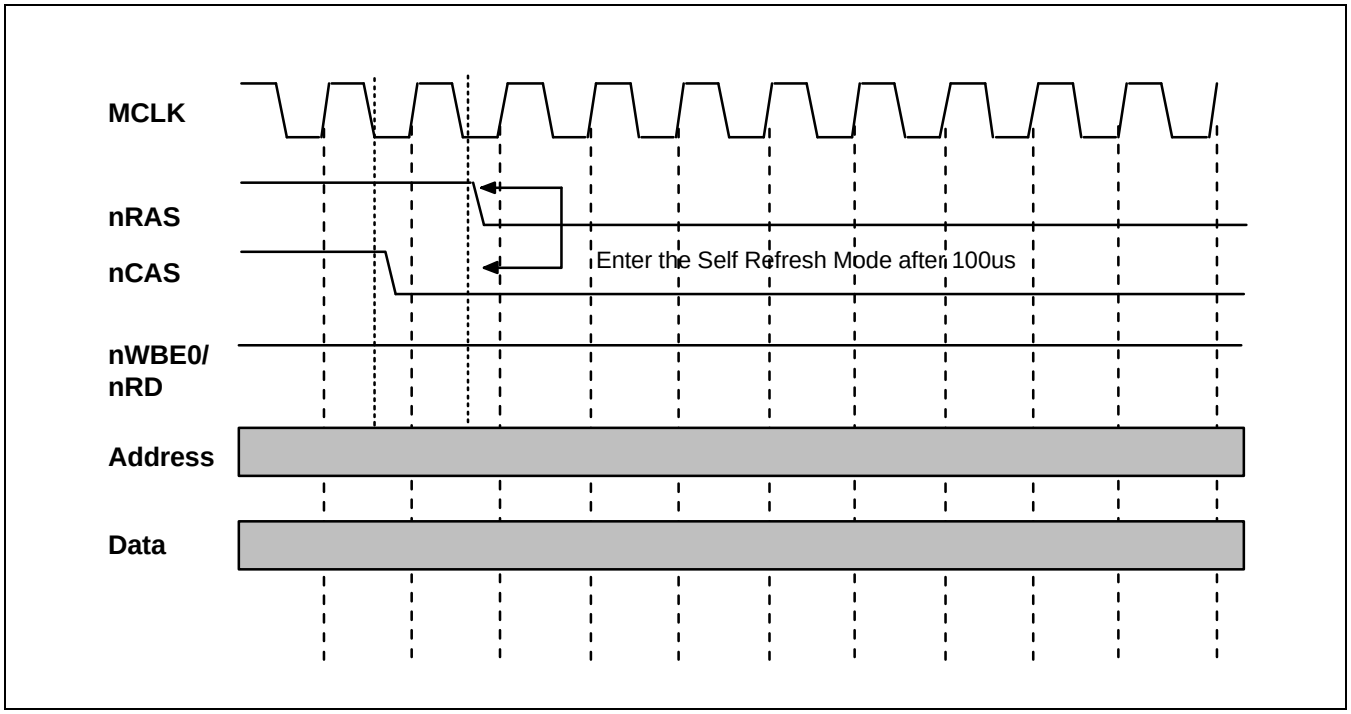


Figure 4-19. DRAM Self Refresh Mode Timing

5 UNIFIED (INSTRUCTION/DATA) CACHE

OVERVIEW

KS17C4000 CPU has a 4-Kbyte internal unified (instruction/data) cache, which adopts an associative two-way set architecture with a four-word (16 bytes) line size. It has a write-through policy to keep data coherency. When cache miss occurs, four words of memory are fetched sequentially from the external memory. It has an LRU (Least Recently Used) algorithm to raise the hit ratio.

The RISC CPU uses the instruction and data cache to improve performance. Without the cache, bottlenecks, which occur during the instruction and data are fetched from the external memory, may seriously degrade performance. The unified cache handles with instruction and data without distinguishing them from one another.

CACHE OPERATION

CACHE ORGANIZATION

The KS17C4000 cache has a 4-Kbyte cache memory and one small Tag RAM. The Tag RAM has a 2-bit CS (Cache Status) and two sets of Tag memories for set 0 and 1. Each Tag set has a 15-bit address field [24:10], which is stored in the cache memory. The 2-bit CS indicates the validity of cached data of the corresponding cache memory line. It is also used for the cache replacing algorithm and for selecting the data coming from Set 0 and 1. Cache memory has two sets, Set 0 and Set 1. Each set has 128-lines and each line has four words of memory space (128-bits).

CACHE REPLACE OPERATION

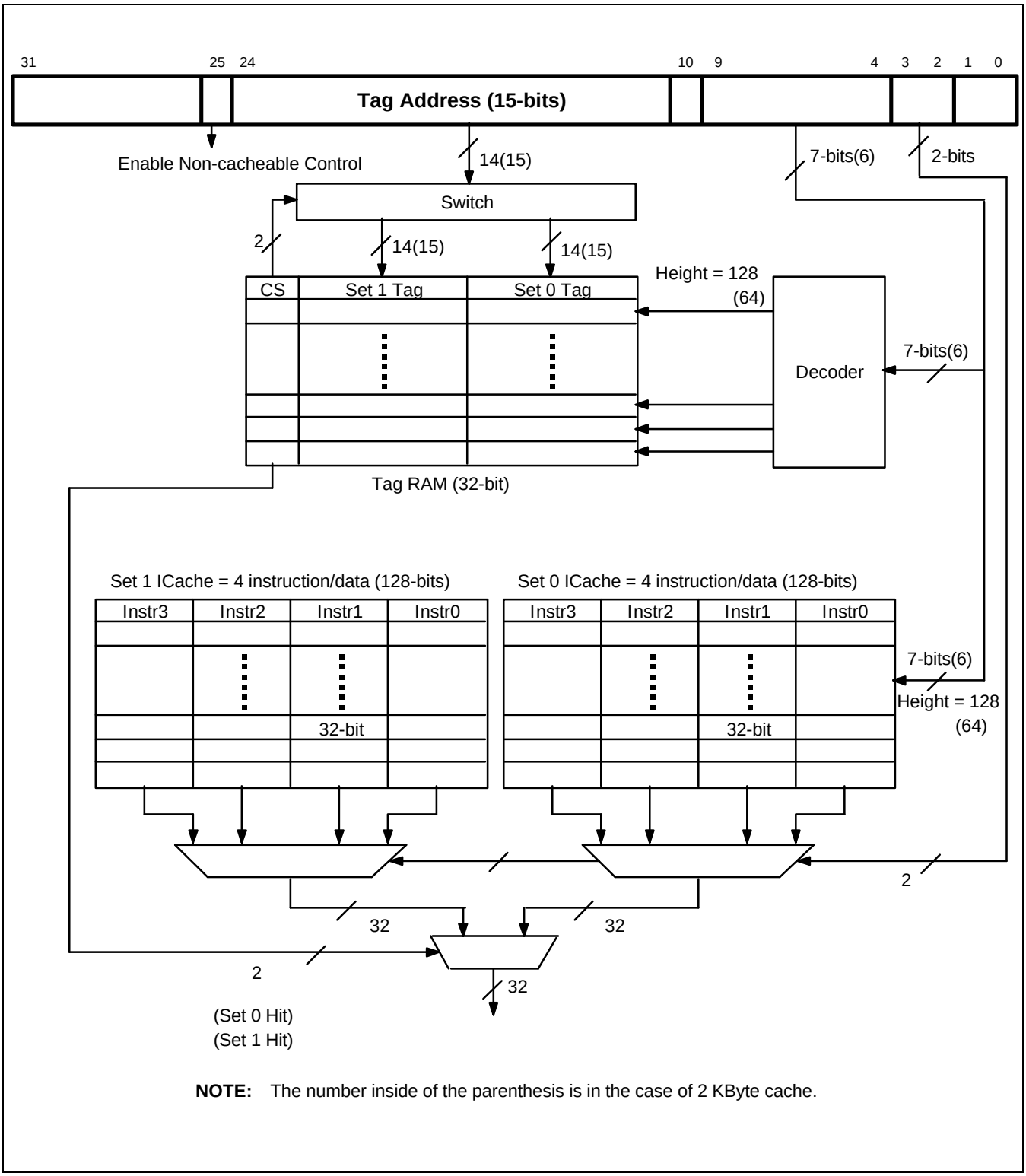
After the system is initialized, the value of CS is set to "00", signifying that the contents of set 0 and set 1 cache memory are invalid. When a cache fill occurs, the value of CS is changed to "01" at the specified line, which signifies that only set 0 is valid. When the subsequent cache fill occurs, the value of CS will be "11" at the specified line, which represents that the contents of both set 0 and set 1 are valid. When the contents of the two sets are valid and the content replacement is required due to the cache miss, the value of CS is changed to "10" at the specified line, signifying that the content of set 0 is replaced. When the value of CS is "10" and another contents replacement is required due to the cache miss, the content of set 1 will be replaced by changing the value of CS to "11".

In conclusion, at a normal steady state, the value of CS will be changed from 11 to 10 (10 to 11), which indicates the information for the implementation of a 2-bit pseudo LRU (Least Recently Used) replacement policy.

CACHE DISABLE OPERATION

The KS17C4000 cache provides programmable entire-cache-enable/disable mode. You can enable the cache by setting the value of CE in SYSCFG to 1, and disable it by clearing SYSCFG[1] to zero. When the cache disable mode is specified, instructions and data are always fetched from the external memory. The KS17C4000 provides a few different cache size, 0KB, 2KB, and 4KB by the Cache Mode bits (SYSCFG[5:4]). The reset SRAM which is not used as the Cache in the 4 KB RAM can be accessed as the internal local 4 KB SRAM. Users can access the internal SRAM from the start address of internal SRAM, which is defined in SYSCFG[15:6].

You have to be cautious about data coherency when the cache memory is enabled again, because the cache memory does not have auto flush mode. You should also be cautious whether or not DMA changes memory data. The DMA accessible memory area should be non-cacheable to keep the data coherency. To keep data coherency between the cache and the external memory, the KS17C4000 uses the write-through method.



NOTE: The number inside of the parenthesis is in the case of 2 KByte cache.

Figure 5-1. Cache Memory Configuration

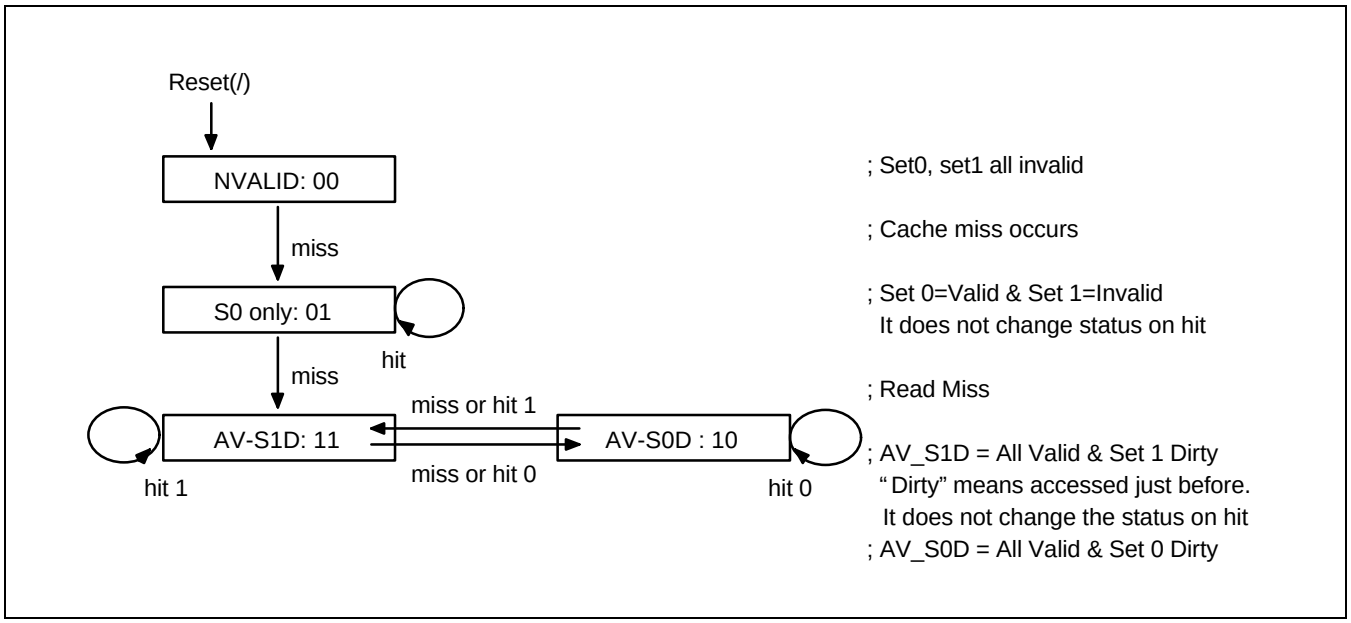
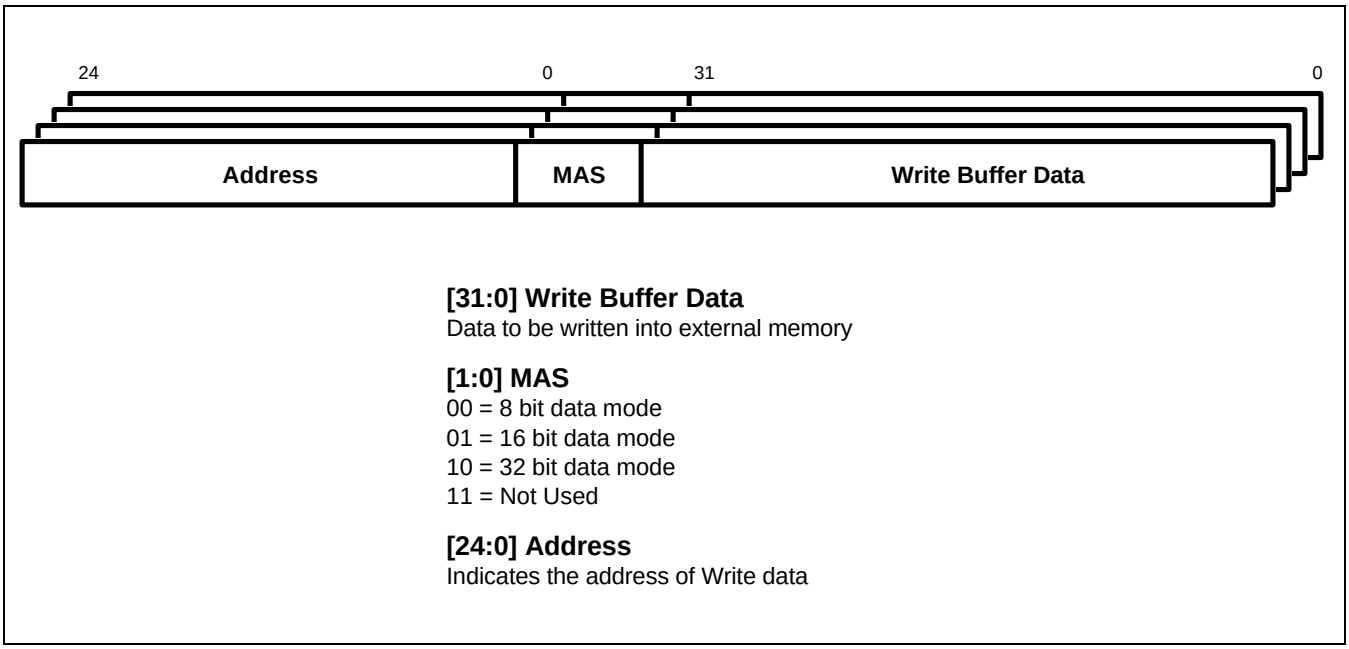


Figure 5-2. CS-Bit Status Diagram

WRITE BUFFER OPERATION

The KS17C4000 has four Write Buffer Registers that enhance memory writing performance. When Write Buffer mode is enabled, the CPU writes data into the write buffer instead of an external memory when the external bus is already occupied by another bus master such as that of DMA. The Write buffer has 4 registers and each register includes a 32-bit data field, a 25-bit address field and a 2-bit status field. The system manager executes all operations of the write buffer



[31:0] Write Buffer Data
Data to be written into external memory

[1:0] MAS
00 = 8 bit data mode
01 = 16 bit data mode
10 = 32 bit data mode
11 = Not Used

[24:0] Address
Indicates the address of Write data

Figure 5-3. Write Buffer Configuration

CACHE FLUSHING

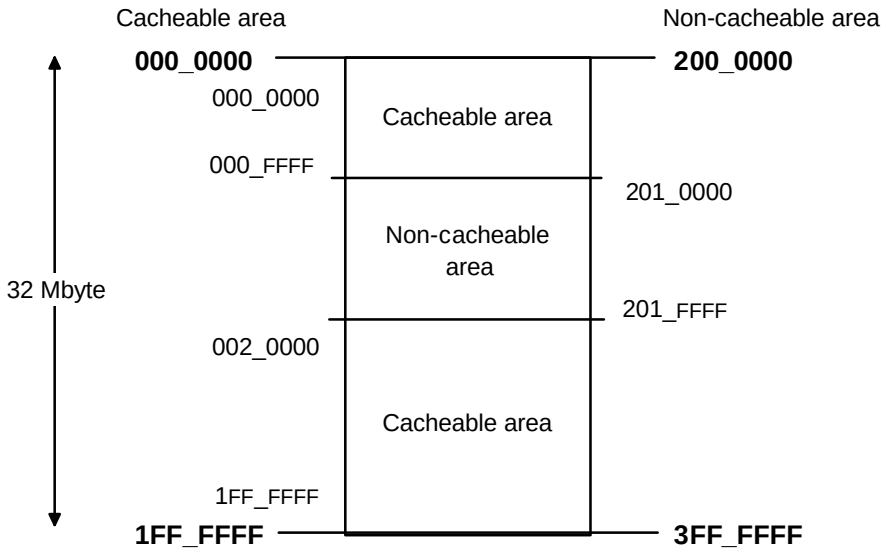
Cache line can be flushed by writing a zero to TAG memory bit 31 and 30, respectively. 2-Kbyte Set 0, 2-Kbyte Set 1 and 512-byte TAG RAM can be accessed from 0x10000000, 0x10800000, and 0x11000000, respectively without setting of the Cache Mode bits and Cache Enable bit. TAG RAM can be cleared by hardware at power on reset. But whenever the Cache or Memory bank configuration is changed on the enable state of cache, TAG RAM must be cleared by user program. A cache flushing is needed to enable the cache operation again. When the cache is disabled, the Tag RAM and cache memory, set 0 and set 1, can be manipulated exactly like in the case of normal memory. You can flush the cache by writing a zero to the Tag RAM and making all the data of the cache invalid. The structure of the Tag RAM and cache memory is shown in Figure 5-1. The memory locations of the Tag RAM and set 0,1 cache memory are as follows:

Item	Address	Comment
Set 0	0x10000000–0x100007ff	(256M) + 2KB
Set 1	0x10800000–0x108007ff	(256M + 8MB) + 2KB
Tag RAM	0x11000000–0x110001ff	(256M + 16M) + 512B

NOTE: Cache flushing must be executed only in the cache disable mode.

NON-CACHE AREA CONTROL BIT

Basically, the cache controls the whole system memory area, but it needs to set non-cacheable areas when the consistency between the memory and the cache should be maintained. The KS17C4000 provides non-cacheable area control bits in address field, ADDR[25]. If the 26th bit of ROM/SRAM/DRAM bank's access address is "0" then the data is cacheable. And If "1", the data accessed is non-cacheable.



NOTE: The non-cacheable area has the same space in memory as the cacheable area. To access the non-cacheable area, you can change the address of the space in memory using non-cacheable control bit.

Figure 5-4. Non-Cacheable Area

6 DMA CONTROLLER

OVERVIEW

The KS17C4000 has two internal direct memory access (DMA) channels that perform the following data transfers without CPU intervention:

— Memory-to-memory

— Serial port-to-memory

— Memory-to-serial port

The on-chip DMA controller can be started by software and/or by an external DMA request (XDREQ). DMA operations can also be stopped and then restarted again by software. The CPU can recognize when a DMA operation has been completed by software polling and/or when it receives an appropriate internally generated DMA interrupt. The KS17C4000 DMA controller can increment or decrement source or destination addresses and conduct 8-bit (byte), 16-bit (half-word), or 32-bit (word) data transfers.

DMA OPERATION

The DMA transfers data directly between a requester and a target. The requester and the target are memory, SIO port, or external devices. An external device requests DMA service by activating XDREQ signals. A channel is programmed by writing to registers which contains the requester address, the target address, the amount of data, and other control contents. SIO, external I/O, or software(memory) can request DMA service. SIO is internally connected to the DMA. Detailed information about the DMA block operation is provided below in the descriptions of the DMA special registers.

There are four data transfer modes: single mode, block mode, four data burst, and demand mode.

DMA SPECIAL REGISTERS

DMA CONTROL REGISTERS (DMACON)

There is a DMA control register in the DMA block.

Table 6-1. DMA Control Registers

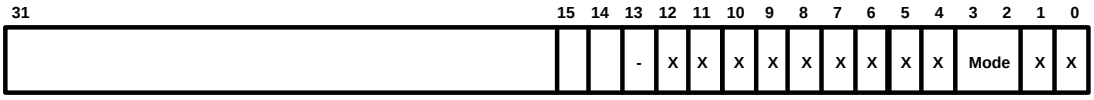
Register	Offset Address	R/W	Description	Reset Value
DMA0CON	0x3000	R/W	DMA 0 control register	0000h
DMA1CON	0x4000	R/W	DMA 1 control register	0000h

The DMACON register contains programmable settings for the following DMA functions.

[0]	Run enable/disable	Normal DMA starts for a specific channel when you set the run bit in the DMA control register. To stop DMA, you must clear the channel run enable/disable bit to 0.
[1]	DMA busy status	When DMA starts, this read-only status bit is automatically set to 1. When it is 0, DMA is in an idle state.
[3:2]	DMA mode selection	Three sources can initiate a DMA operation: software, an external DMA request (nXDREQ), and the SIO (UART or Sync. SIO) block. The mode selection setting determines which source can initiate a DMA operation at any given time (see Figure 6-1).
[4]	Destination address direction	This bit controls whether the destination address will be decremented or incremented during a DMA operation.
[5]	Source address direction	This bit controls whether the source address will decrement or increment during a DMA operation.
[6]	Destination address fix	This bit determines whether the destination address will be changed or not during a DMA operation. This feature is used when transferring data from multiple sources to a single destination.
[7]	Source address fix	This bit determines whether the source address will be changed or not during a DMA operation. This feature is used when transferring data from a single source to multiple destinations.
[8]	DMA stop interrupt enable	A DMA operation is started/stopped by setting/clearing the run enable/disable bit. If this bit is set to 1 and DMA starts, a stop interrupt is generated when the DMA operation stops. If this bit is 0, an interrupt is not generated.
[9]	4 data burst mode (1)	If this bit is set to 1, then 4 data burst mode will be selected.
[10]	SIO transfer direction	This setting is for serial I/O operations only. When bit 10 is set to 1, the direction of serial data transfer is memory-to-SIO. When it is 0, the direction is SIO-to-memory.

[11]	Single/block mode	This bit determines the number of external DMA requests nXDREQs) that are required for a DMA operation when DMA is configured for external DMA (DMAnCON[3:2] = 0,1): When DMAnCON[11] = 1 in block mode, the KS17C4000 requires only one DMA request during the entire DMA operation. When DMAnCON[11] is 0, the KS17C4000 requires one external DMA request per single DMA operation. A single DMA operation is defined as a single (one-time) DMA read and write cycle. In other words, this control bit determines how many DMA requests are needed to complete an entire DMA operation.
[13:12]	Transfer width	To transfer data in half word (16-bit) units, you set these bits to 01. For byte-wise DMA transfers, set those bits to 00. For word-wise DMA transfers, set those bits to 10.
[14]	Continuous mode	This bit specifies whether a DMA operation holds the system bus or not until the count value becomes 0. Therefore, this bit must be carefully used for the whole operation time not to exceed the appropriate interval. (Ex. DRAM refresh)
[15]	Demand mode	To run the external DMA operation so fast, set this bit. If this bit is set during the DMA operation, DMA never goes to the idle state. Altogether, the external device transfers/receives the mount of data which it wants to transfer/receive. The amount of data depends on how long XDREQ signal is active.

NOTE: If a DMA is set as four data burst and continuous mode together, 4 burst mode is ignored, and the continuous mode only is operated. In order to use 4 burst mode in DMA operation, please be sure that continuous mode is disabled.



[0] Run enable bit
0 = Disable DMA operation
1 = Enable DAM operation

[1] DMA busy state indicator
0 = DMA idle
1 = DAM active

[3:2] DMA mode selection bits
00 = Software
01 = External nXDREQ
10 = UART
11 = Sync. SIO

[4] Destination address direction
0 = Increase destination address during DMA
1 = Decrease destination address during DMA

[5] Source address direction
0 = Increase source address during DMA
1 = Decrease source address during DMA

[6] Destination address fix
0 = Change destination address during DMA
1 = Do not change destination address during DMA

[7] Source address fix
0 = Change source address during DMA
1 = Do not change sour ce address during DMA

[8] DMA stop interrupt enable
0 = Do not generate stop interrupt when DMA stops
1 = Generate stop interrupt when DMA stops

[9] 4 burst mode
0 = Normal operation
1 = 4 burst operating mode

[10] Transfer direction (for serial I/O only)
0 = SIO-to-memory
1 = Memory-to-SIO

[11] Single/block mode (nXDREQs)
0 = One nXDREQ initiates a single DMA operation
1 = One nXDREQ initiates a whole DMA operation

[13:12] DMA transfer width
00 = Byte transfer
01 = Hlaf-word transfer
10 = Word transfer

[14] Continuoud mode (CN)
0 = Normal operation mode
1 = Hold system bus until the whole DMA stops

[15] Demand mode (DM)
0 = Normal external DMA mode
1 = Demand mode

Figure 6-1. DMA Control Registers (DMA0CON, DMA1CON)

DMA SOURCE/DESTINATION ADDRESS REGISTERS

There are DMA source and destination address registers in the DMA block. DMAnSRC contains the 25-bit source start address for a DMA channel. DMAnDST contains the 25-bit destination start address for a DMA channel.

Table 6-2. DMA Source/Destination Address Registers

Register	Offset Address	R/W	Description	Reset Value
DMA0SRC	0x3004	R/W	DMA 0 source address register	000 0000h
DMA0DST	0x3008	R/W	DMA 0 destination address register	000 0000h
DMA1SRC	0x4004	R/W	DMA 1 source address register	000 0000h
DMA1DST	0x4008	R/W	DMA 1 destination address register	000 0000h

Depending on the settings you make to the DMA control register (DMAnCON), the source/destination address stored in the DMAnSRC/DMAnDST register will be incremented or decremented in byte or half word units during a DMA operation.

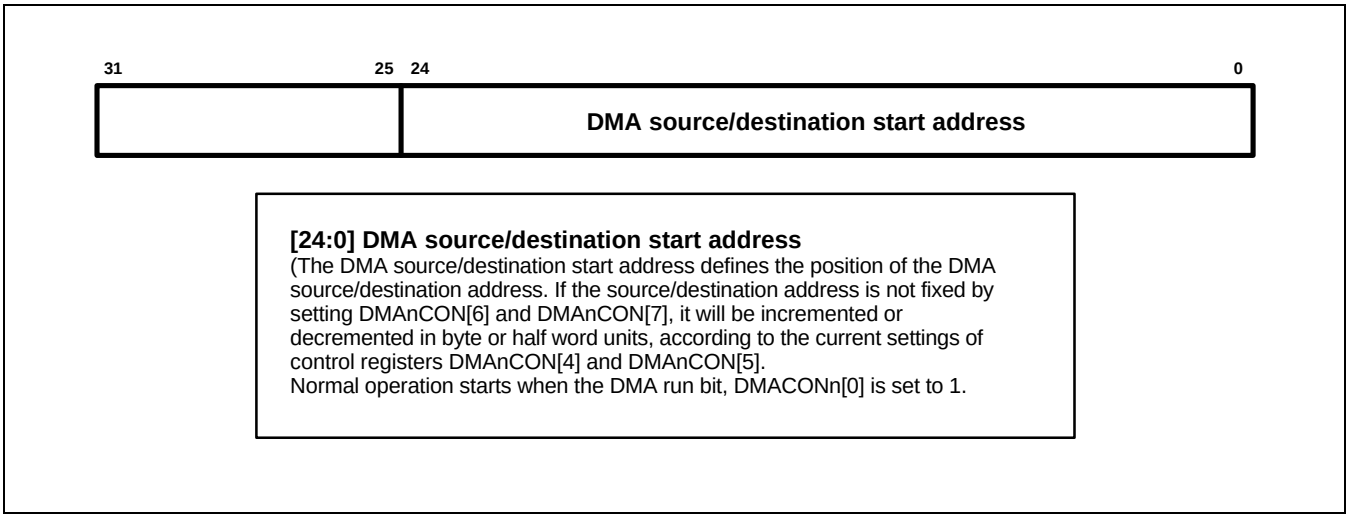


Figure 6-2. DMA Source/Destination Address Registers (DMASRC/DMADST)

DMA TRANSFER COUNT REGISTER

A DMA transfer count register in the DMA block, DMAnCNT, contains the current count value of the number of DMA transfers completed for a DMA channel. See Figure 6-3 below.

Table 6-3. DMA Transfer Count Registers

Register	Offset Address	R/W	Description	Reset Value
DMA0CNT	0x300c	R/W	DMA 0 transfer count register	00 0000h
DMA1CNT	0x400c	R/W	DMA 1 transfer count register	00 0000h

If you initialize the transfer count register value to 0xffffh, the maximum transfer count in words or in bytes can be calculated as (64 Kbytes)/(64 K half-words). In this case, a DMA operation decrements the transfer count register value by one.

A special feature of the KS17C4000 DMA controller is that you can also write the value 0x0000h to the DMA count register field. With this value, the number of DMA transfers is counted as 16 Mbytes/16 M half words.

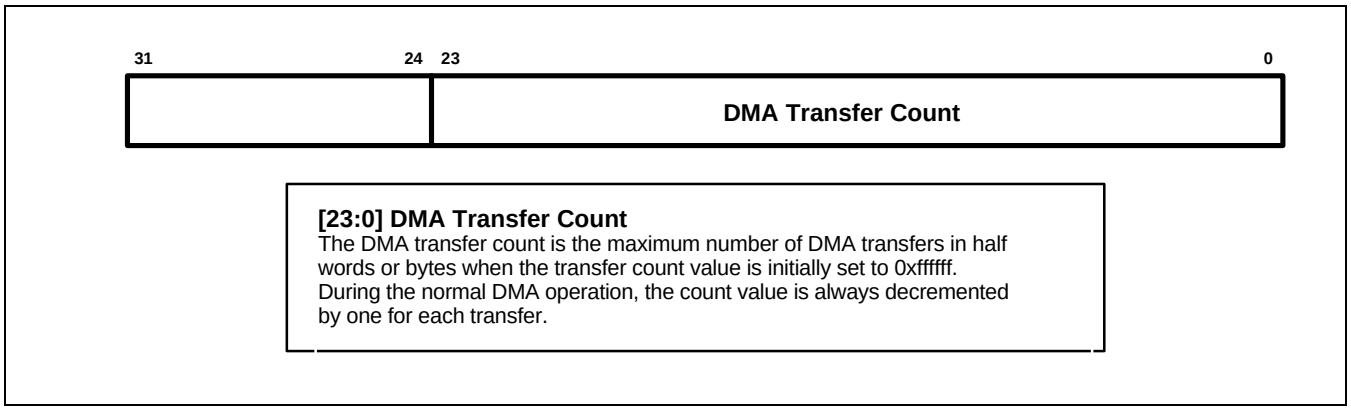


Figure 6-3. DMA Transfer Count Registers (DMACNT)

DMA FUNCTION DESCRIPTION

The following sections provide a functional description of the DMA controller operations.

DMA TRANSFERS

The DMA transfers data directly between a requester and a target. The requester and target are memory,UART or external devices. An external device requests DMA service by activating nXDREQ signal. A channel is programmed by writing to registers which contain requester address, target address, the amount of data, and other control contents. UART, external I/O, or Software(memory) can request DMA service. UART is internally connected to the DMA.

STARTING/ENDING DMA TRANSFERS

DMA starts to transfer data after the DMA receives service request from nXDREQ signal, UART, or Software. When the entire buffer of data has been transferred, the DMA becomes idle. If you want to preform another buffer transfer, the DMA must be reprogrammed. Although the same buffer transfer wii be preformed again, the DMA must be reprogrammed.

DATA TRANSFER MODES

Single Mode

A DMA request (nXDREQ or an internal request) causes one byte, one half-word, or one word to be transmitted if 4-data burst mode is disable state, or four times of transfer width if 4-data burst mode is enable state. Single mode requires a DMA request for each data transfer. The nXDREQ signal can be de-asserted after checking that nXDACK has been asserted.

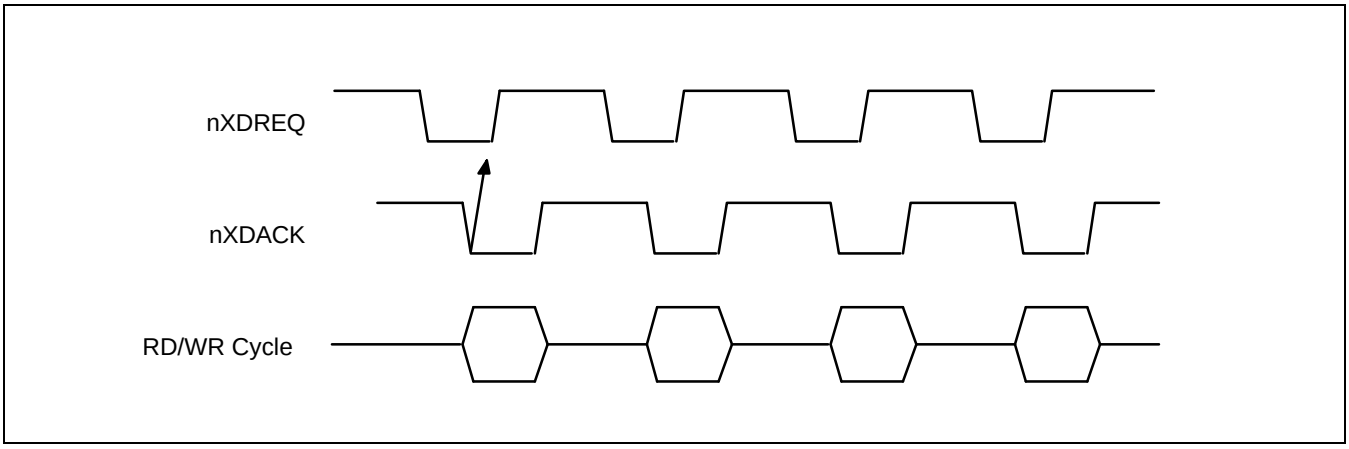


Figure 6-4. External DMA Requests (Single Mode)

Block (Continuous) Mode

The assertion of only one DMA request (nXDREQ or an internal request) causes all of the data, as specified by the control register settings, to be transmitted in a single operation. The DMA transfer is completed when the transfer counter value reaches zero. The nXDREQ signal can be de-asserted after checking that nXDACK has been asserted.

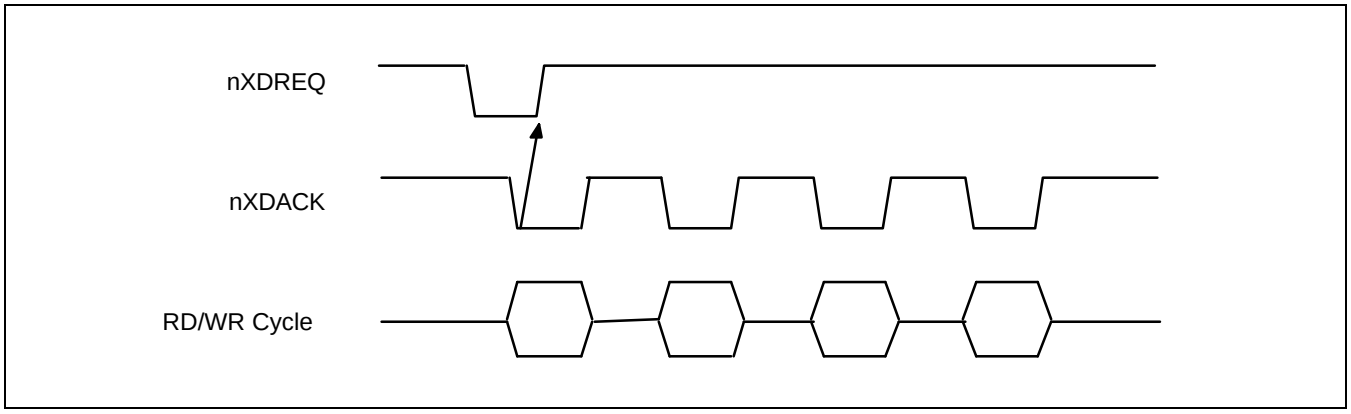


Figure 6-5. External DMA Requests (Block Mode)

Demand Mode

In demand mode, the DMA continues transferring data as long as the DMA request input (nXDREQ) is held active.

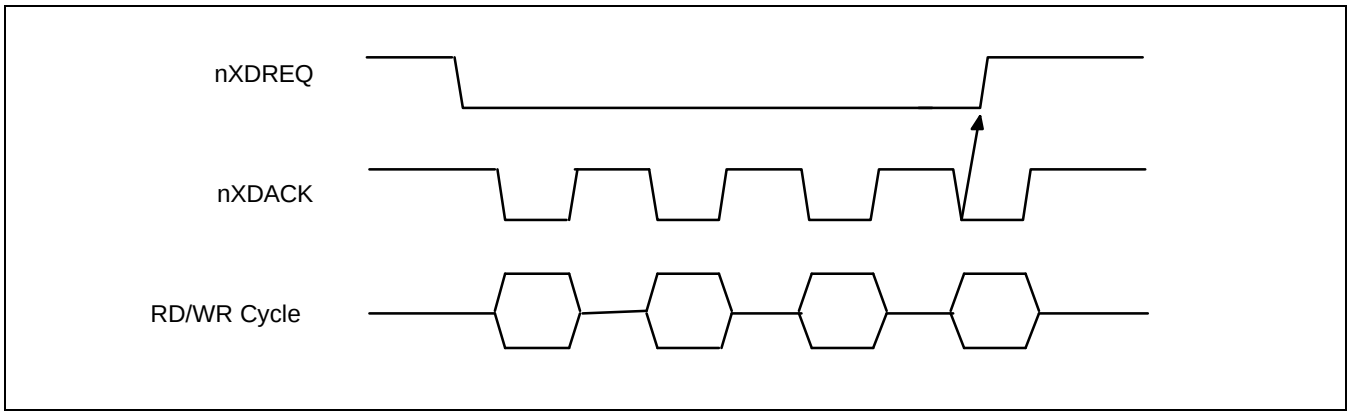


Figure 6-6. External DMA Requests (Demand Mode)

DMA TRANSFER TIMING DATA

Figure 6-6 provides detailed timing data for DMA data transfers that are triggered by external DMA requests. Please note that read/write timing depends on which memory banks are selected.

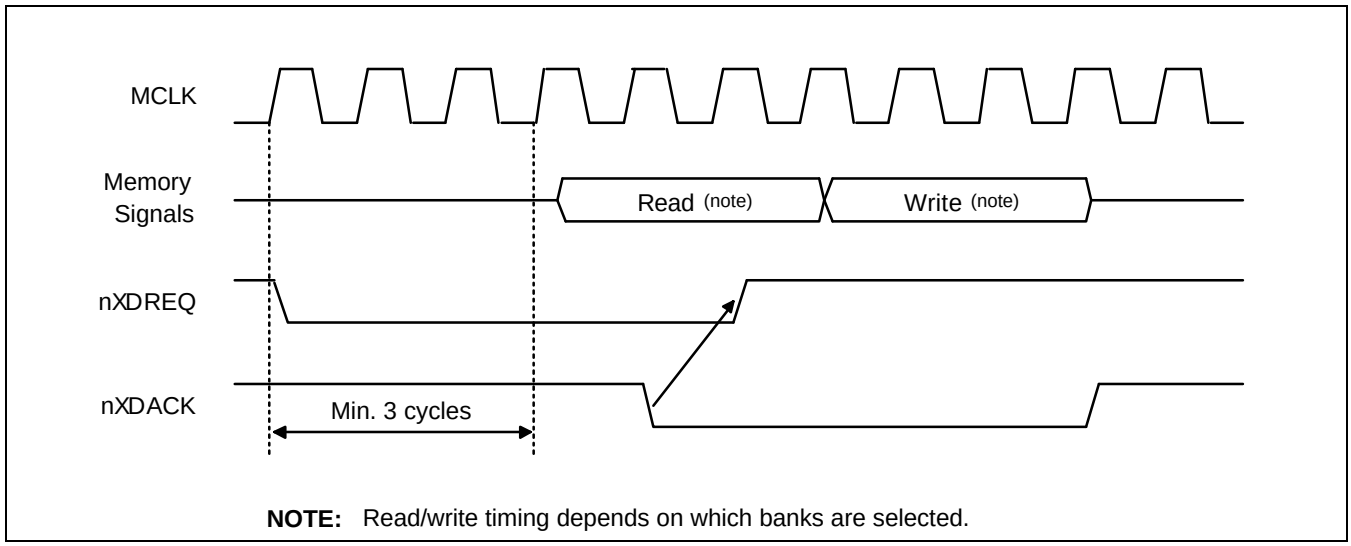


Figure 6-7. External DMA Requests Detailed Timing

NOTES

7 SERIAL I/O

OVERVIEW

The KS17C4000 has an on-chip UART (Universal Asynchronous Receiver/Transmitter) block and another on-chip synchronous serial I/O ports 0/1. SIO blocks can be operated in interrupt-based, UART and synchronous SIO operation of SIO 0/1 can be operated in DMA-based mode. That is, both UART and SIO 0/1 can generate a DMA request for data transfer.

SIO functions as a universal asynchronous receiver and transmitter (UART) and synchronous serial I/O 0/1 with three wires to communicate external device.

UART

An UART contains a programmable baud rate generator, Rx and Tx port for UART communication, Tx and Rx shift registers, Tx and Rx buffer registers, Tx and Rx control blocks and control registers. Important features of the UART block include programmable baud rates, transmit/receive (full duplex mode), one or two stop bit insertion, 5-bit, 6-bit, 7-bit, or 8-bit data transmit/receive, and parity checking.

The baud rate generator can be clocked by the internal oscillation clock. The transmitter and the receiver contain Tx and Rx data buffer register and a Tx and an Rx shift register respectively. Data to be transmitted is written to the Tx buffer register, then copied to the Tx shift register, and shift out by the transmit data pin (Tx). Data received is shifted in by the receive data pin (Rx), then copied from shift register to the Rx buffer register whenever one data byte is received. The control unit provides control for mode selection and status/interrupt generation.

UART Baud rate = Source clock / (16 x (Pre-scaler Value + 1))

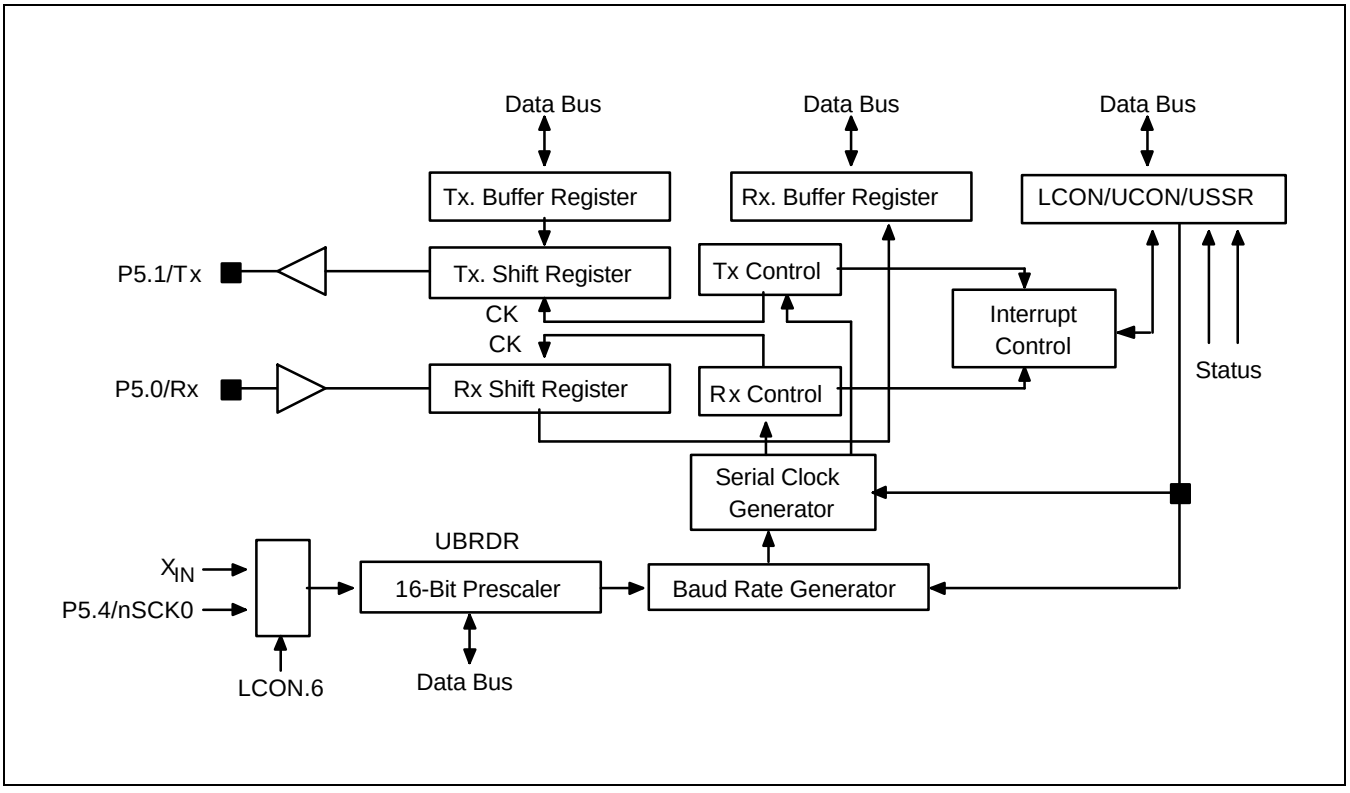


Figure 7-1. UART Block Diagram

INTERRUPT/DMA REQUEST GENERATION

KS17C4000 UART has seven status signals: overrun error, parity error, frame error, break, receive buffer full, transmit buffer register empty and transmitter empty, all of which are indicated by the corresponding UART status register (USSR).

The overrun error, parity error, frame error and break condition are referred to as the receive status, each of which can cause the receive status interrupt request, if the receive-status-interrupt-enable bit is set to one in the control register UCON. When a receive-status-interrupt-request is detected, you can know the signal which causes the request by reading the status register (USSR).

When the receiver transfers the data of the receive shifter to the receive buffer register, it activates the receive buffer full status signal which will cause the receive interrupt, if the receive mode in control register is selected as the interrupt mode. When the transmitter transfers data from its transmit buffer register to its shifter, the transmit holding register empty status signal is activated. The signal causes the transmit interrupt if the transmit mode in control register is selected as interrupt mode.

The receive-buffer-full and transmit-buffer-register empty status signals can also be connected to generate the DMA request signals if the receive/transmit mode in the control register is selected as the DMA mode.

As mentioned before, two DMA channels, DMA0 and DMA1, are provided in the KS17C4000.

LOOPBACK MODE

The KS17C4000 UART provides a test mode referred as the loopback mode to aid in isolating faults in the communication link. In this mode, the transmitted data is immediately received. This feature allows the processor to verify the internal transmit and to receive the data path of UART channel. This mode can be selected by setting the loopback-bit in the UART control register (UCON).

DATA TRANSMISSION

The data frame for transmission is programmable. It consists of a start bit, 5 to 8 data bits, an optional parity bit and 1 to 2 stop bits, which can be specified by the line control register (LCON). The transmitter can also produce the break condition. The break condition forces the serial output to logic 0 state for a duration longer than one frame transmission time. At the receiving end, the break condition sets an error flag as mentioned above.

The data transmission process is shown in Figure 7-2. The transmitter transfers data through a path as follows: data source -> transmit buffer register -> transmit shifter -> Tx pin. It then completes the parallel-to-serial data conversion. Two flags (status signals), transmit buffer register empty and transmitter empty, are used to indicate the status of the transmit buffer register and transmitter.

DATA RECEPTION

Like the transmission, the data frame for reception is also programmable. It consists of a start bit, 5 to 8 data bits, an optional parity bit and 1 to 2 stop bits by settings in the line control register (LCON). The receiver can detect overrun error, parity error, frame error and break condition, each of which can set an error flag.

The overrun error indicates that new data has overwritten the old data before the old data has been read. The parity error indicates that the receiver has detected a parity condition other than what it was programmed for. The frame error indicates that the received data does not have a valid stop bit. The break condition indicates that the Rx input is held in the logic 0 state for a duration longer than one frame transmission time.

The data reception process is shown in Figure 7-2. The receiver transfers data through a path as follows: Rx pin -> receive shift register -> receive buffer register -> destination. This completes the serial-to-parallel data conversions. In addition to receive-error-status flags, a receive-buffer-full flag is used to indicate the status of the receive buffer register.

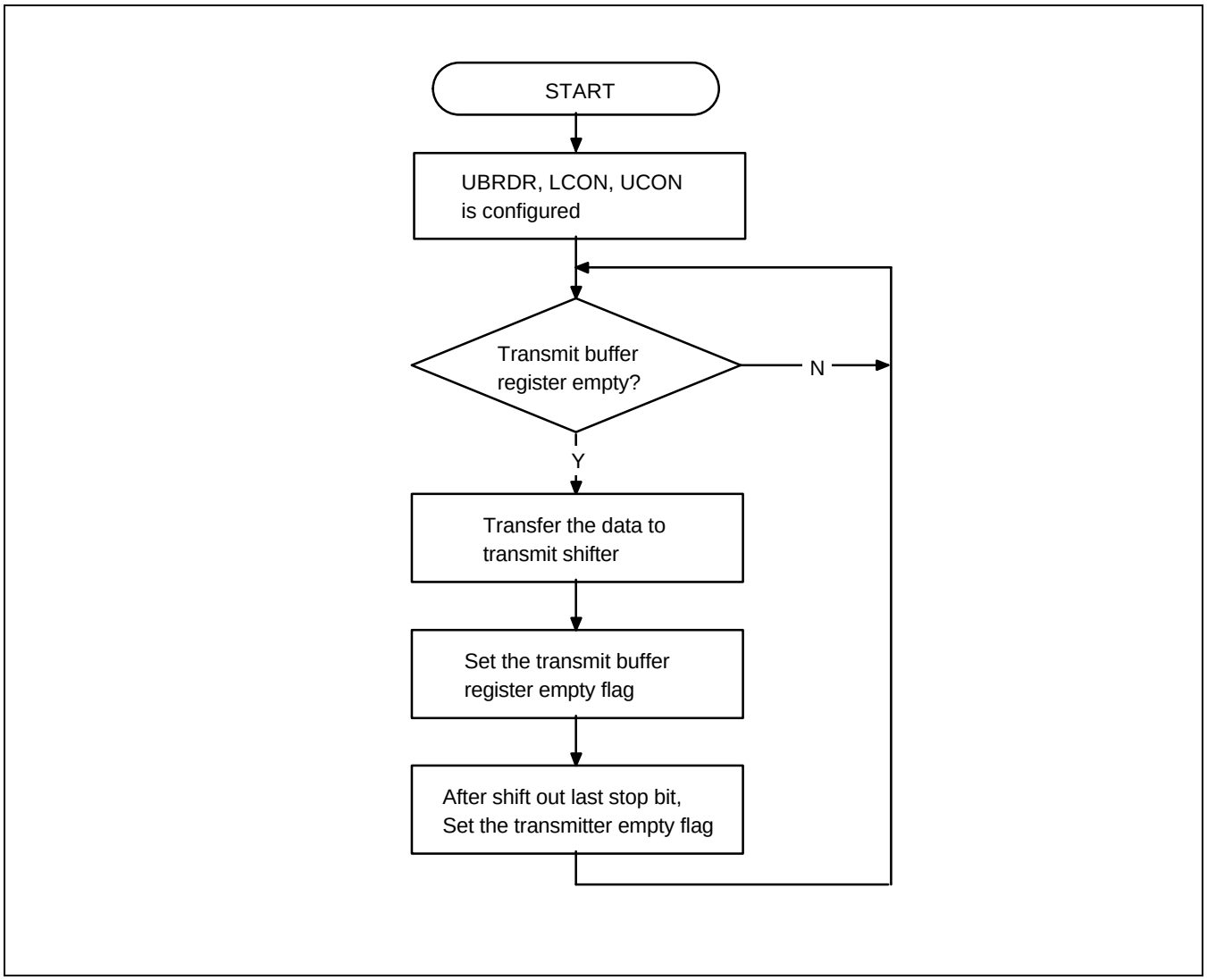


Figure 7-2. UART Data Transmission Process

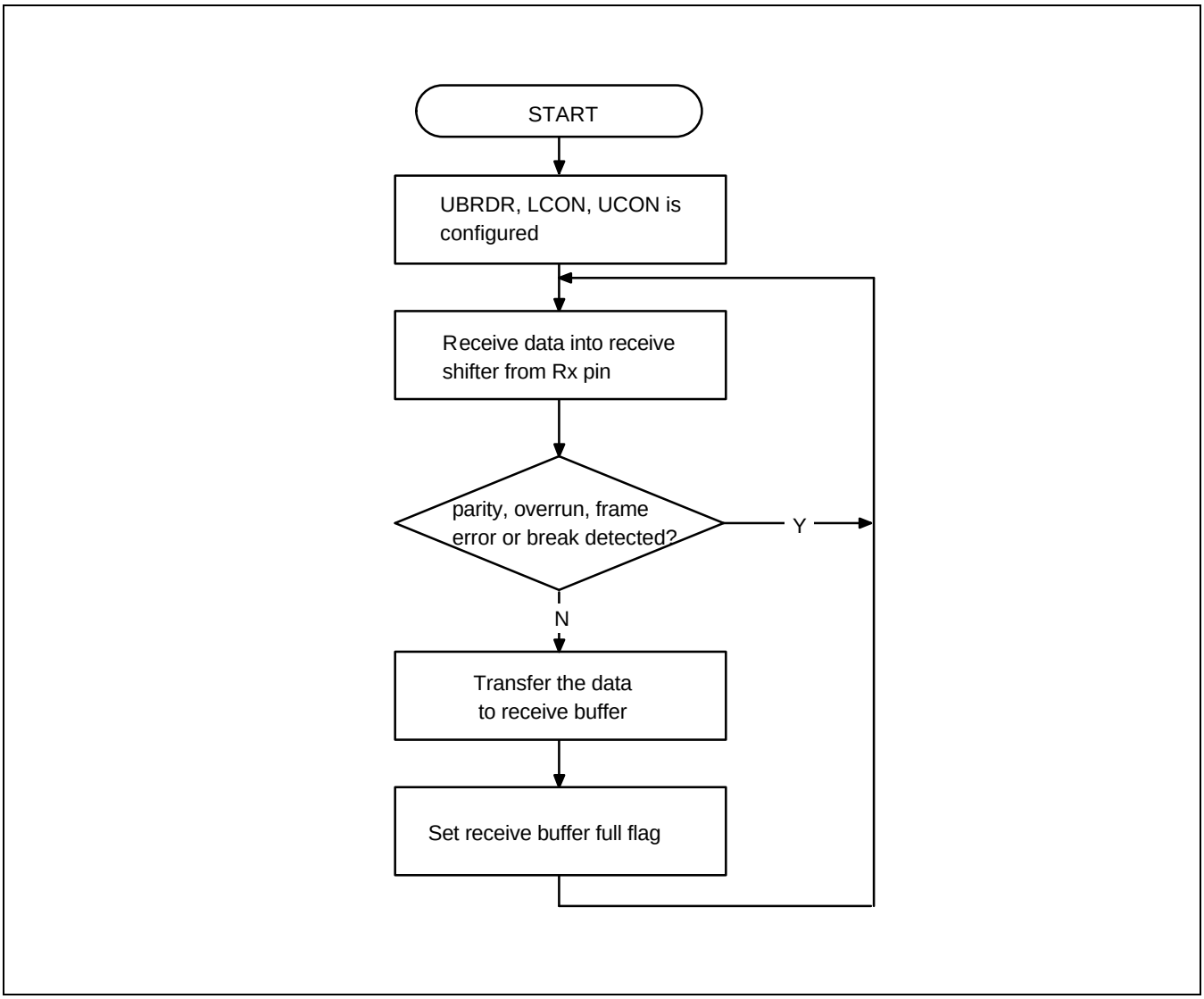


Figure 7-3. UART Data Reception Process

UART SPECIAL REGISTERS

UART LINE CONTROL REGISTER

The UART Line control register, LCON, is used to control the UART.

Register	Offset Address	R/W	Description	Reset Value
LCON	0x5003	R/W	UART line control register	00h

[1:0]	Word length (WL)	The 2-bit word length value indicates the number of data bits to be transmitted or received per frame. The options are 5-bit, 6-bit, 7-bit, or 8-bit.
[2]	Number of stop bits	LCON[2] specifies how many stop bits are used to signal end-of-frame (EOF). When it is 0, one bit signals EOF; when it is 1, two bits signal EOF.
[5:3]	Parity mode (PMD)	The 3-bit parity mode value specifies how parity generation and checking are to be performed during UART transmit and receive operations. There are five options (see Figure 7-2).
[6]	Baud rate clock selection	When LCON[6] is 0, the internal system clock (Xin) is selected as the baud rate generator clock source. When it is 1, an external clock source[nSCK0] is selected as the baud rate generator clock source.

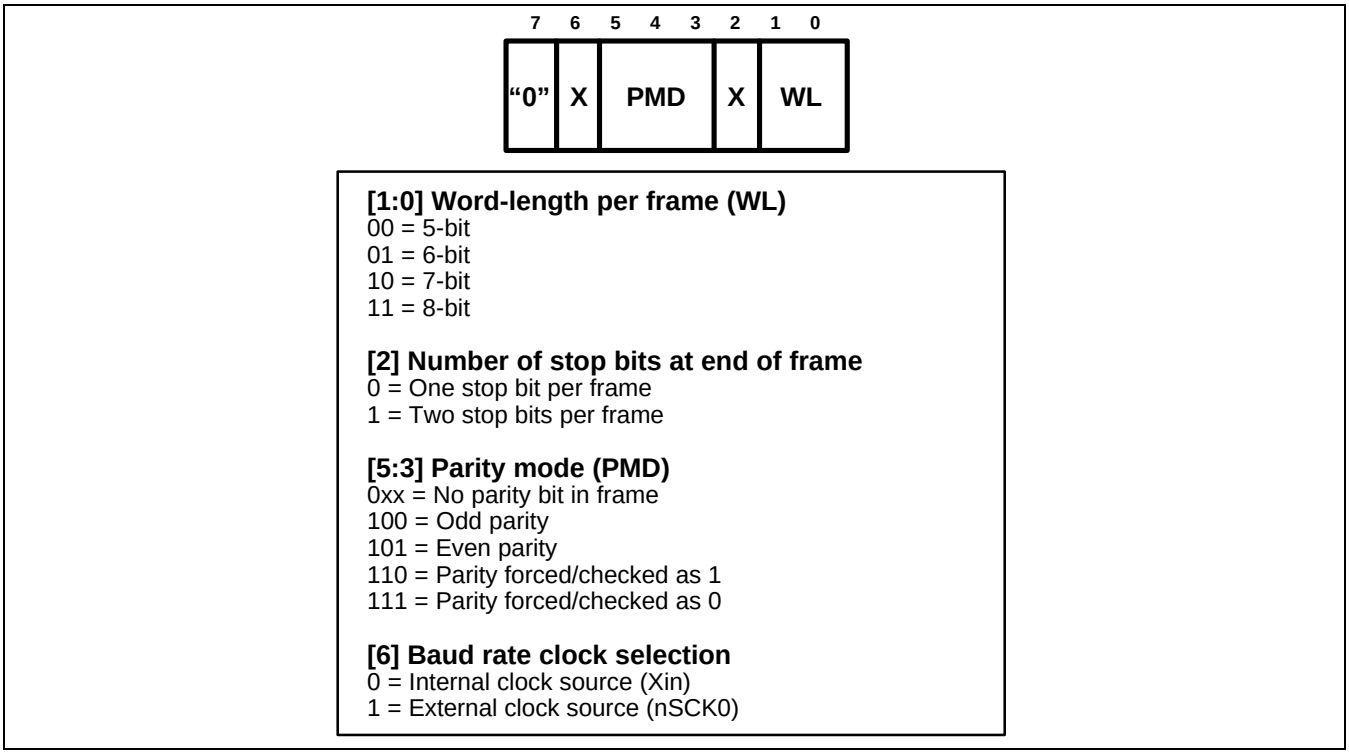


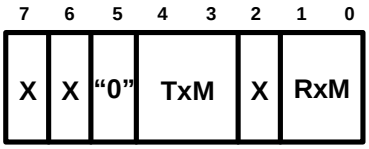
Figure 7-4. UART Line Control Register (LCON)

UART CONTROL REGISTER

The UART control register, UCON, is used to control the single-channel UART.

Register	Offset Address	R/W	Description	Reset Value
UCON	0x5007	R/W	UART control register	00h

[1:0]	Receive mode (RxM)	This two-bit value determines which function is currently able to read data from the UART receive buffer register, RBR. The three options are interrupt routine, or DMA channel.
[2]	Rx status interrupt enable	This bit enables the UART to generate an interrupt if an exception (break, frame error, parity error, or overrun error) occurs during a receive operation. When UCON[2] is set to 1, a receive status interrupt will be generated each time an Rx exception occurs. When UCON[2] is 0, no receive status interrupt will be generated.
[4:3]	Transmit mode (TxM)	This two-bit value determines which function is currently able to write Tx data to the UART transmit buffer register, TBR. The options are interrupt routine, or DMA channel.
[6]	Send break	Setting UCON[6] causes the UART to send a break. A break is defined as a continuous Low level signal on the transmit data output with a duration of more than one frame transmission time. By setting this bit when the transmitter is empty (transmitter empty bit, USSR[7] = 1), you can use the transmitter to time the frame. When USSR[7] is 1, write the transmit buffer register, TBR, with the data to be transmitted, And then poll the USSR[7] value. When it returns to 1, clear (reset) the send break bit, UCON[6].
[7]	Loopback bit	Setting UCON[7] causes the UART to enter loopback mode. In loopback mode, the transmit data output is sent High level and the transmit buffer register (TBR) is internally connected to the receive buffer register (RBR). This mode is provided for test purposes only.



[1:0] UART receive mode selection (RxM)
00 = Disable
01 = Interrupt request
10 = DMA0 request
11 = DMA1 request

[2] Receive status interrupt enable
0 = Do not generate receive status interrupt
1 = Generate receive status interrupt

[4:3] UART transmit mode selection (TxM)
00 = Disable
01 = Interrupt request
10 = DMA0 request
11 = DMA1 request

[6] Send break
0 = Do not send break
1 = Send break

[7] Loopback enable
0 = Normal UART operating mode
1 = Enable UART loopback mode (for testing only)

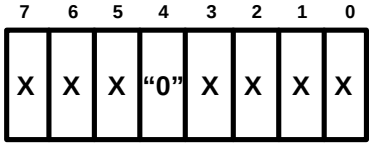
Figure 7-5. UART Control Register (UCON)

UART STATUS REGISTER

The UART status register, USSR, is a read-only register that is used to monitor the status of serial I/O operations in the single-channel UART.

Register	Offset Address	R/W	Description	Reset Value
USSR	0x500b	R	UART status register	c0h

[0]	Overrun error	USSR[0] is automatically set to 1 whenever an overrun error occurs during a serial data receive operation. When the receive status interrupt enable bit, UCON[2] is 1, a receive status interrupt will be generated if an overrun error occurs. This bit is automatically cleared to 0 whenever the UART status register (USSR) is read.
[1]	Parity error	USSR[1] is automatically set to 1 whenever a parity error occurs during a serial data receive operation. When the receive status interrupt enable bit, UCON[2] is 1, a receive status interrupt will be generated if a parity error occurs. This bit is automatically cleared to 0 whenever the UART status register (USSR) is read.
[2]	Frame error	USSR[2] is automatically set to 1 whenever a frame error occurs during a serial data receive operation. When the receive status interrupt enable bit, UCON[2] is 1, a receive status interrupt will be generated if a frame error occurs. The frame error bit is automatically cleared to 0 whenever the UART status register (USSR) is read.
[3]	Break interrupt	USSR[3] is automatically set to 1 to indicate that a break signal has been received. If the receive status interrupt enable bit, UCON[2], is 1, a receive status interrupt will be generated if a break occurs. The break interrupt bit is automatically cleared to 0 when you read the UART status register.
[5]	Receive data ready	USSR[5] is automatically set to 1 whenever the receive data buffer register (RBR) contains valid data received over the serial port. The data received can then be read from the RBR. When this bit is 0, the RBR does not contain valid data. Depending on the current setting of the UART receive mode bits, UCON[1:0], an interrupt or a DMA request is generated when USSR[5] is 1.
[6]	Tx buffer register empty	USSR[6] is automatically set to 1 when the transmit buffer register (TBR) does not contain valid data. In this case, the TBR can be written with the data to be transmitted. When this bit is 0, the TBR contains valid Tx data that has not yet been copied to the transmit shift register. In this case, the TBR cannot be written with new Tx data. Depending on the current setting of the UART transmit mode bits, UCON[4:3], an interrupt or a DMA request will be generated when USSR[6] is 1.
[7]	Transmitter empty (T)	USSR[7] is automatically set to 1 when the transmit buffer register has no valid data to transmit and when the Tx shift register is empty. When the transmitter empty bit is 1, it indicates to software that it can now disable the transmitter function block.



[0] Overrun error
0 = No overrun error during receive
1 = Overrun error (generate receive status interrupt, if UCON[2] is 1)

[1] Parity error
0 = No parity error during receive
1 = Parity error (generate receive status interrupt, if UCON[2] is 1)

[2] Frame error
0 = No frame error during receive
1 = Frame error (generate receive status interrupt, if UCON[2] is 1)

[3] Break interrupt
0 = No break received
1 = Break received (generate receive status interrupt, if UCON[2] is 1)

[5] Receive data ready
0 = No valid data in the receive buffer register
1 = Valid data present in the receive buffer register (issue interrupt or DMA request, if UCON[1:0] is set)

[6] Transmit holding register empty
0 = Valid data in transmit holding register
1 = No data in transmit holding register (issue interrupt or DMA request, if UCON[4:3] is set)

[7] Transmitter empty
0 = Transmitter not empty; Tx in progress
1 = Transmitter empty; no data for Tx

Figure 7-6. UART Status Register (USSR)

UART TRANSMIT BUFFER REGISTER

The UART transmit holding register, TBR, contains an 8-bit data value to be transmitted over the single-channel UART.

Register	Offset Address	R/W	Description	Reset Value
TBR	0x500f	W	Serial transmit buffer register	xxh

[7:0]	Transmit data	This field contains the data to be transmitted over the single-channel UART. When this register is written, the transmit buffer register empty bit in the status register, USSR[6], should be 1. This prevents overwriting transmit data that may already be present in the TBR. Whenever the TBR is written with a new value, the transmit register empty bit, USSR[6], is automatically cleared to 0.

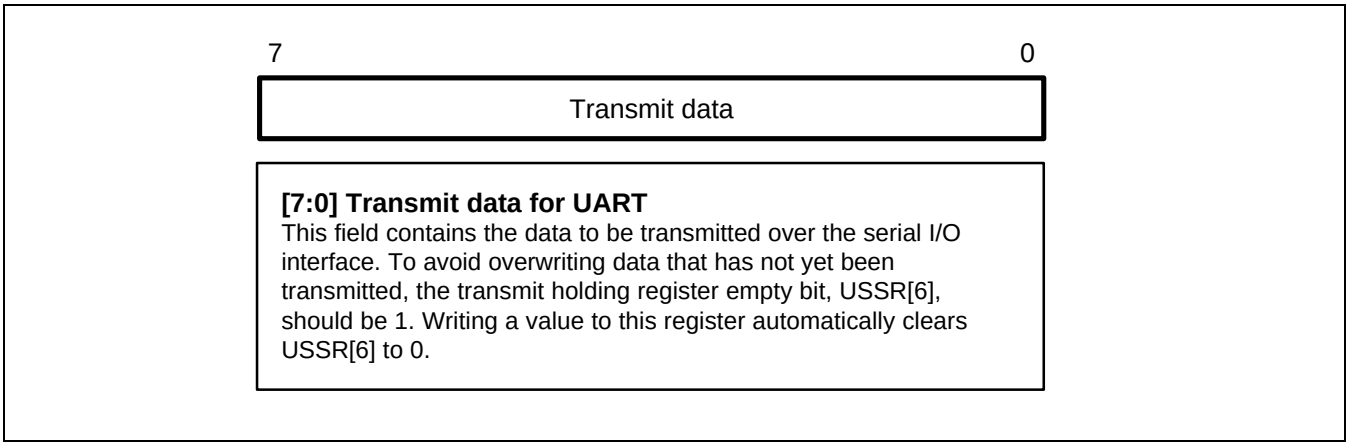


Figure 7-7. UART Transmit Buffer Register (TBR)

UART RECEIVE BUFFER REGISTER

The receive buffer register, RBR, contains an 8-bit field for received serial data.

Register	Offset Address	R/W	Description	Reset Value
RBR	0x5013	R	Serial receive buffer register	xxh

[7:0] Receive data

This field contains the data received over the single-channel UART. When this register is read, the receive data ready bit in the UART status register, USSR[5], should be 1. This prevents reading invalid receive data that may already be present in the RBR. Whenever the RBR is read, the receive data ready bit, USSR[5], is automatically cleared to 0.

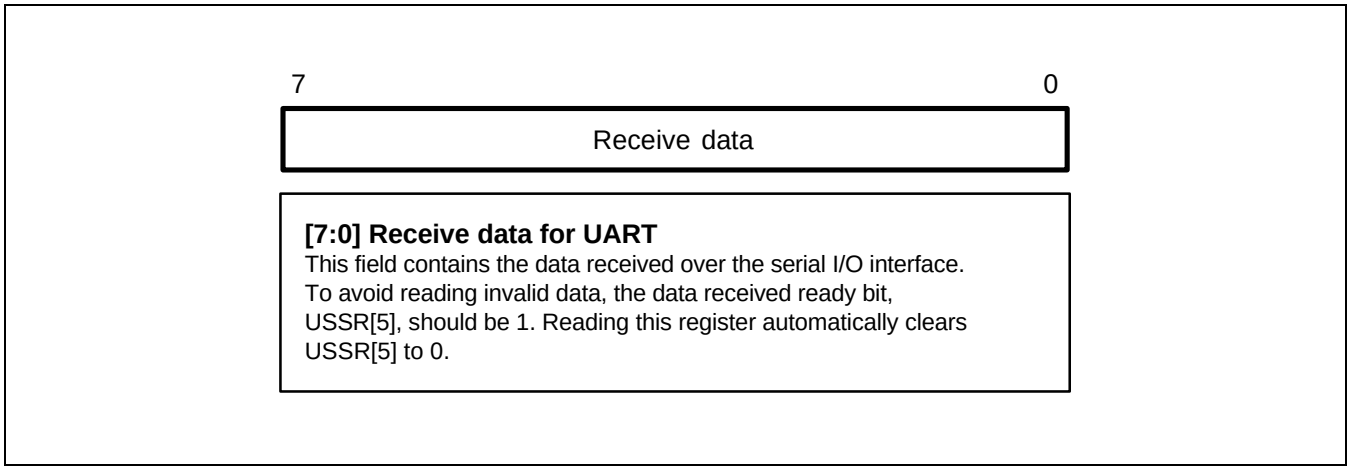


Figure 7-8. UART Receive Buffer Register (RBR)

UART BAUD RATE PRESCALER REGISTERS

The value stored in the baud rate divisor register, UBRDR, is used to determine the serial Tx/Rx clock rate (baud rate) as follows:

Baud rate = source_clock / ((Divisor value + 1) x 16)

The source_clock is either MCLK (the internal master clock, Xin) or nSCK0 (the external UART clock input, P5.4/nSCK0) and it is determined by the setting of the serial clock selection bit in the line control register, LCON[6].

Register	Offset Address	R/W	Description	Reset Value
UBRDR	0x5016	R/W	Baud rate divisor register	0000h

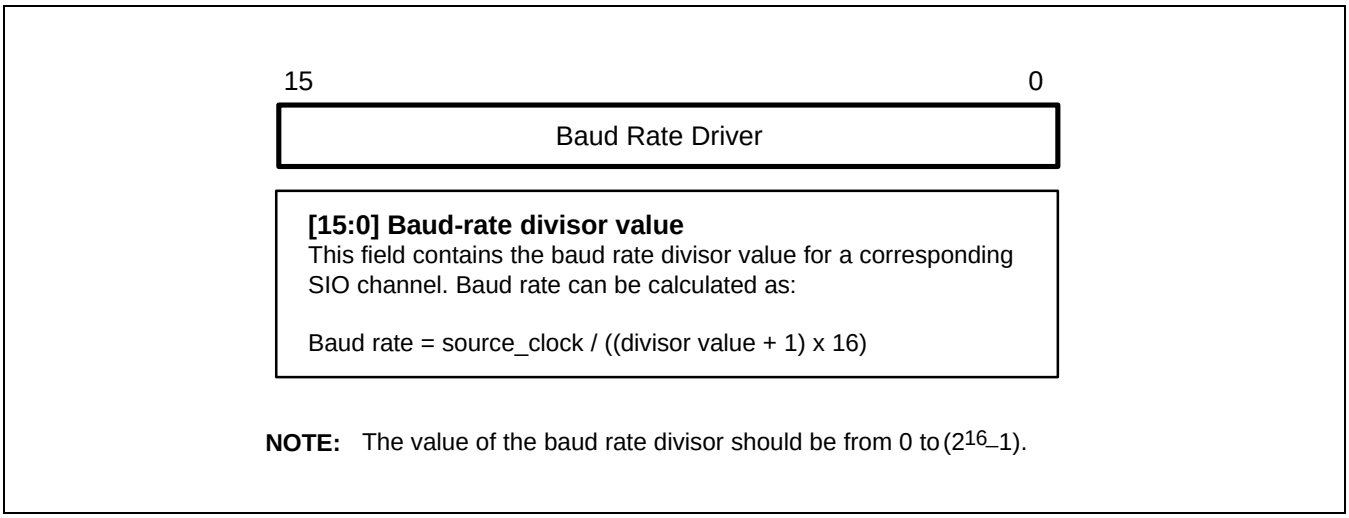


Figure 7-9. UART Baud Rate Divisor Registers (UBRDR)

UART BAUD RATE COUNT AND CLOCK REGISTERS

For test or specfic purposes only, the internal baud rate up counters, BRDCNT, can be directly accessed using register addressing. In addition, the baud rate clock can be monitored through the UART Tx port, Tx/P5.1.

If the BRCKCON monitor value is "1", the baud rate clock can be monitored at the Tx/P5.1 pin. If it is "0" (its default value), or if you write a "0" to the BRCKCON address, the UART Tx data signal output to the Tx/P5.1 depends on the current setting of Port 5 control register bit 2, 3, and 4.

Table 7-1. BRDCNT and BRCKCON Registers

Register	Offset Address	R/W	Description	Reset Value
BRDCNT	0x501a	W	UART baud rate count register	0x0
BRCKCON	0x501f	W	UART baud rate clock control register	0x0

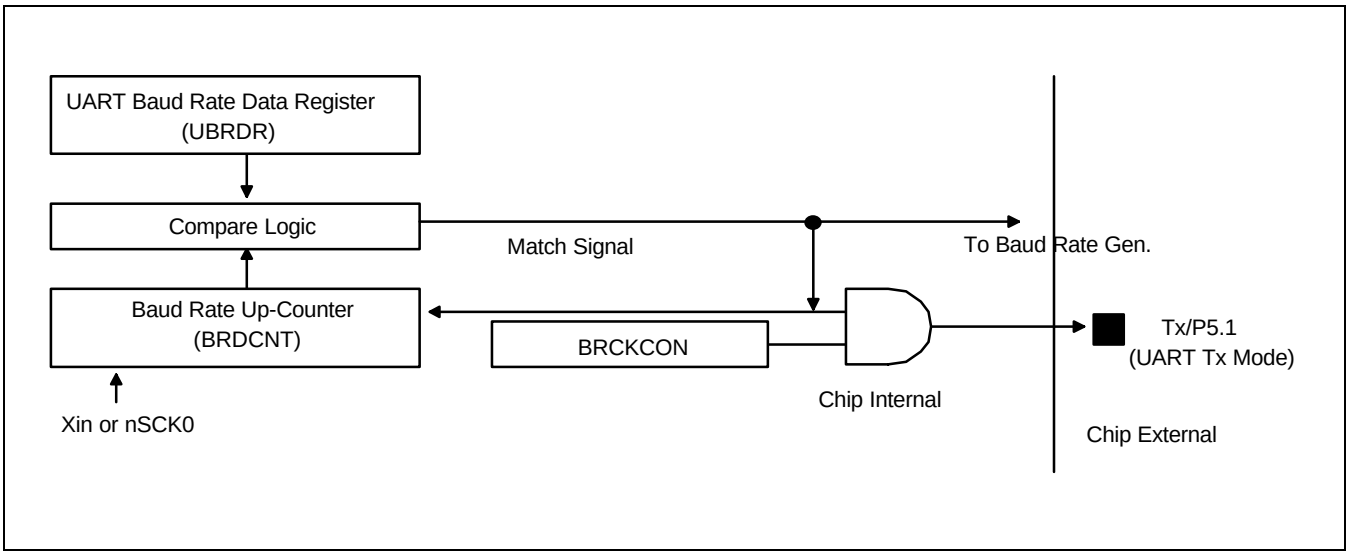


Figure 7-10. UART Baud Rate Clock Test Scheme

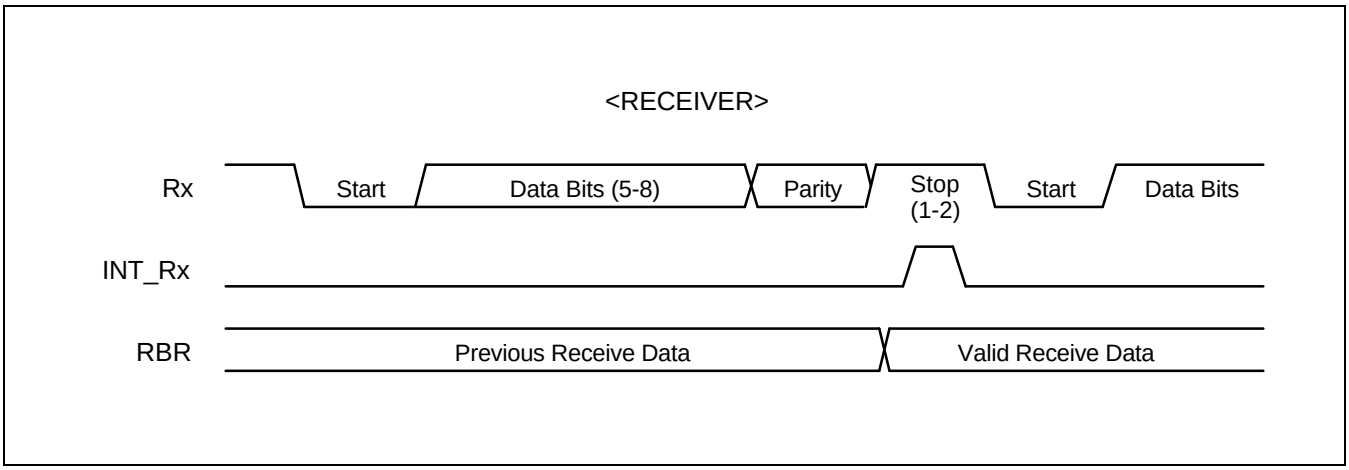


Figure 7-11. Interrupt-Based Serial I/O Timing Diagram (Rx Only)

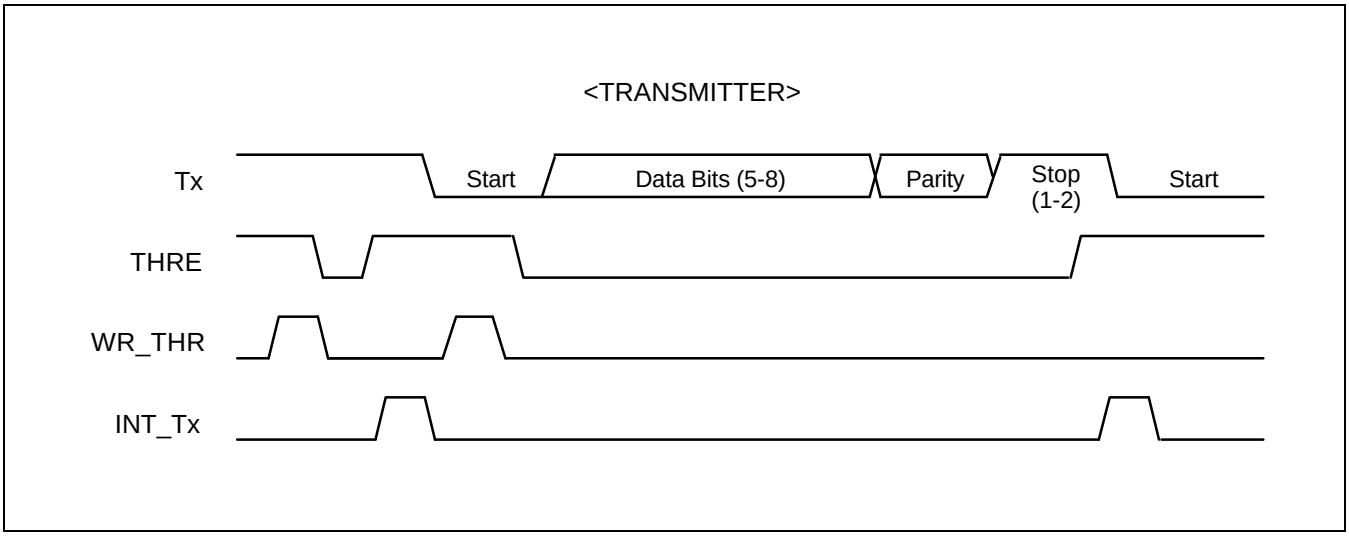


Figure 7-12. Interrupt-Based Serial I/O Timing Diagram (Tx Only)

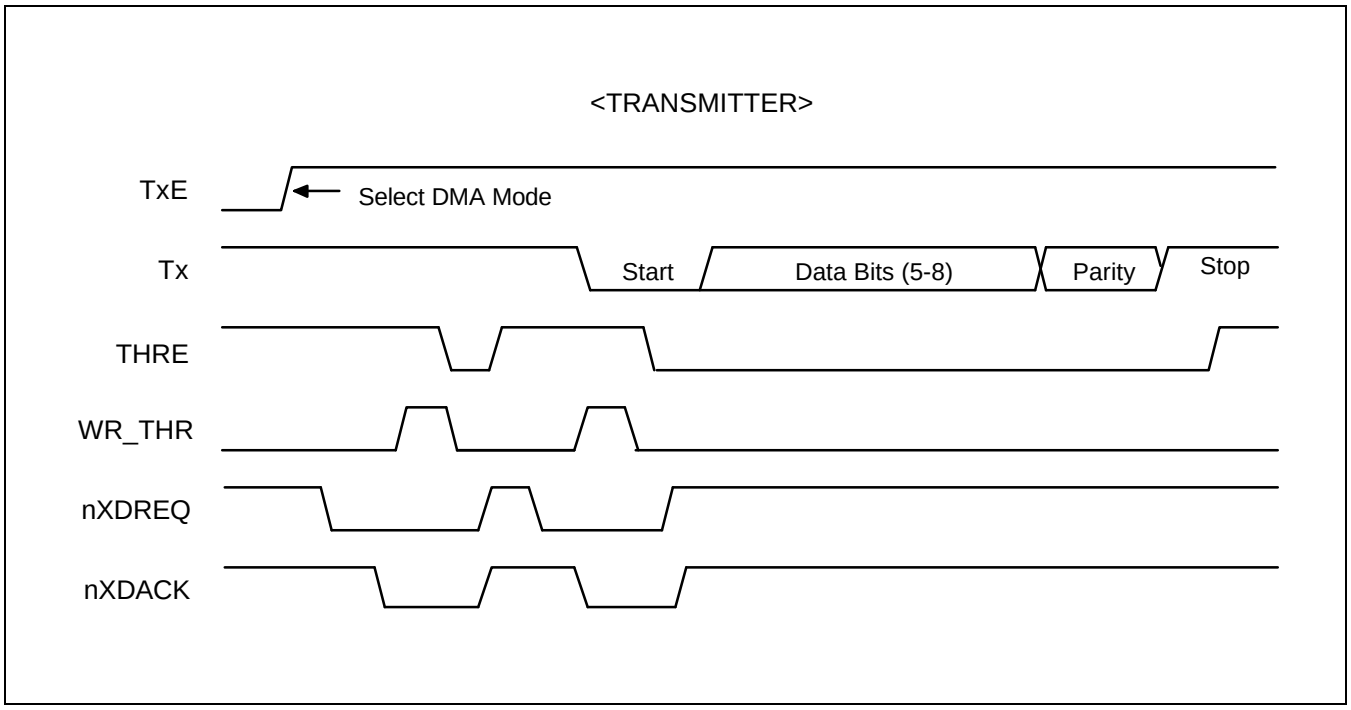


Figure 7-13. DMA-Based Serial I/O Timing Diagram (Tx Only)

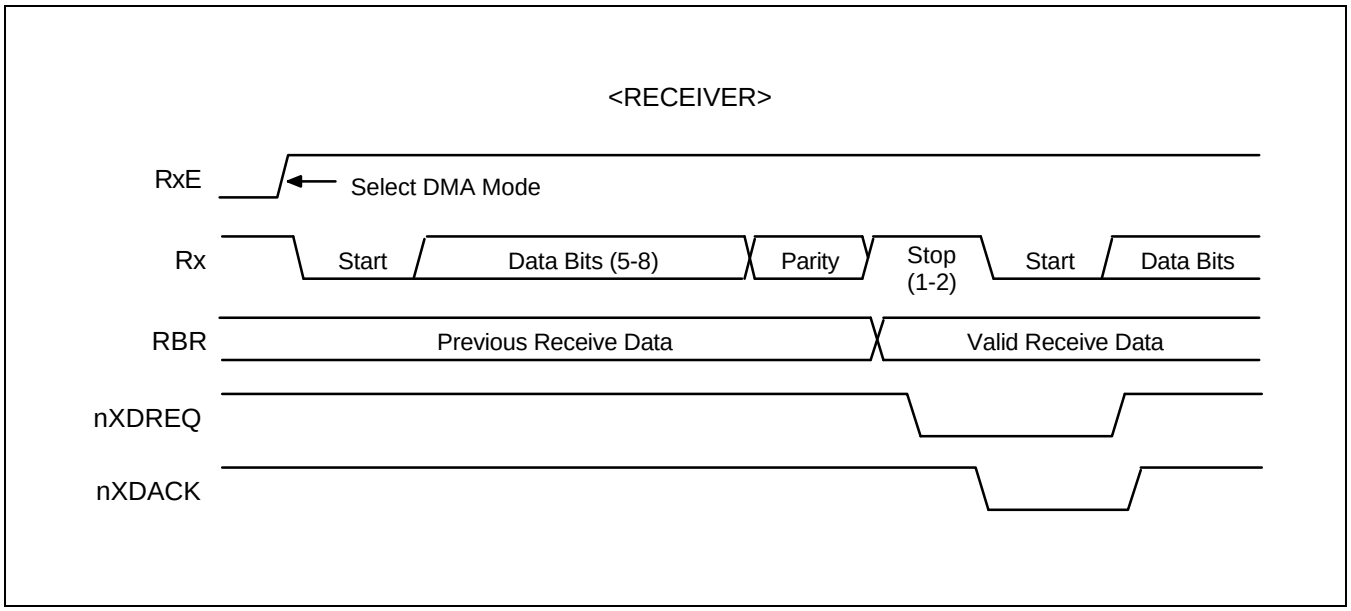


Figure 7-14. DMA-Based Serial I/O Timing Diagram (Rx Only)

SYNCHRONOUS SIO 0 AND 1 (SYNCHRONOUS SIO 0/1)

OVERVIEW

Serial I/O 0/1 module, SIO 0/1 can interface with various types of external devices that require a serial data transfer. The components of each SIO function block are:

— 8-bit control register (SIOnCON)
— Clock selection logic
— 8-bit data buffer (SIOnDATA)
— 8-bit prescaler (SBRDRn)
— 8-bit byte interval counter for DMA mode (BIDRn)
— 3-bit serial clock counter
— Serial data I/O pins (SIn, SOn)
— External clock input/out pin (SCKn)

The SIO module can transmit or receive 8-bit serial data at a frequency determined by the setting of the corresponding control register. To ensure flexible data transmission rates, you can select an internal or external clock source.

PROGRAMMING PROCEDURE

To program the SIO modules, follow these basic steps:

1. Configure the I/O pins at port (SO, SCK, SI) by loading the appropriate value to the P5CON register, if necessary.
2. Load an 8-bit value to the SIOnCON control register to properly configure the serial I/O module. In this operation, SIOnCON.2 must be set to "1" to enable the data shifter.
3. For interrupt generation, set the serial I/O interrupt enable bit (IntMask.17 or 18) to "1".
4. When you transmit data to the serial buffer, write data to SIOnDATA and set SIOnCON.3 to 1. Then, the shift operation starts.
5. When the shift operation (transmit/receive) is completed, the SIO pending bit (IntPend.17 or 18) is set to "1" and an SIO interrupt request is generated.

Using the serial I/O interface, 8-bit data can be exchanged with an external device. The transmission frequency is controlled by making the appropriate bit settings to the SIO0CON/SIO1CON register and BRDR0/BRDR1 register. The serial interface can run an internal or an external clock source. If an internal clock signal is used, you can modify its frequency to adjust the baud rate divisor register value. When the programmer writes a byte data to SIO0DATA/SIO1DATA register, SIO0/1 transmit operation will start, if SIO0/1 Run bit is set and Tx mode bit is enabled.

SIO 0/1 CONTROL REGISTER(SIO0CON/SIO1CON)

The control register for serial I/O interface module, SIO0CON/SIO1CON, is located at 0x6003/0x7003. It has the control settings for the SIO module.

— Clock source selection (internal or external) for shift clock
— Select interrupt or DMA interrupt.
— Edge selection for shift operation
— Clear 3-bit counter and start shift operation
— Hand shake mode selection for the DMA operation.
— Mode selection (transmit/receive or receive-only)
— Data direction selection (MSB first or LSB first)

A reset clears the SIOnCON value to '00H'. This configures the corresponding module with an external clock source at the SCK, selects receive-only operating mode, and clears the 3-bit counter. The data shift operation and interrupts are disabled. The selected data direction is MSB-first.

The serial I/O port control register, SIO0CON/SIO1CON, is used to control the SIO 0/1.

Register	Offset Address	R/W	Description	Reset Value
SIO0CON	0x6003	R/W	Serial I/O port 0 control register	00h
SIO1CON	0x7003	R/W	Serial I/O port 1 control register	00h

[1:0]	SIO mode selection	This two-bit value determines which function is currently able to read/write data from/to SIODATA. 00: No interrupt pending 01: SIO interrupt 10: DMA0 request 11: DMA1 request
[2]	Hand-shaking enable	This bit value determines whether hand shaking operation is enabled or disabled when DMA mode is operated.
[3]	SIO Start	This bit value determines whether SIO function run and counter is cleared or stopped.
[4]	Clock Phase selection	This bit determines whether serial transmit or receive. Operation synchronizes with a rising or falling edge of the clock respectively. 0: Transmit action at falling, receive at rising edge of clock 1: Transmit action at rising, receive at falling edge of clock
[5]	SIO Tx enable	This bit decides whether the transmit operation is enabled or not. 0: Receive-only mode 1: Tranmit/Receive mode
[6]	MSB/LSB selection	This bit controls whether MSB is first transmitted or LSB is first transmitted. When SIOnCON[6] is 0, Most Significant Bit (MSB) is transmitted first. When SIOnCON[6] is 1, Least Significant Bit (LSB) is transmitted first.

[7]	Shift clock selection	When SIOnCON[7] is 0, the baud rate generator clock is selected as the shift clock source. When it is 1, an external clock source is selected as the shift clock source.

When SIOn is set at synchronous SIO mode, the serial output data goes out through a serial output pin (SOn), and the serial input data come in through a serial input pin (SIn), by transferring one bit by one serial input/output data synchronously with the serial clock pin (SCKn). After transmitting a data or receiving data, the SIO interrupt request is activated unless programmer does not enable the interrupt source.

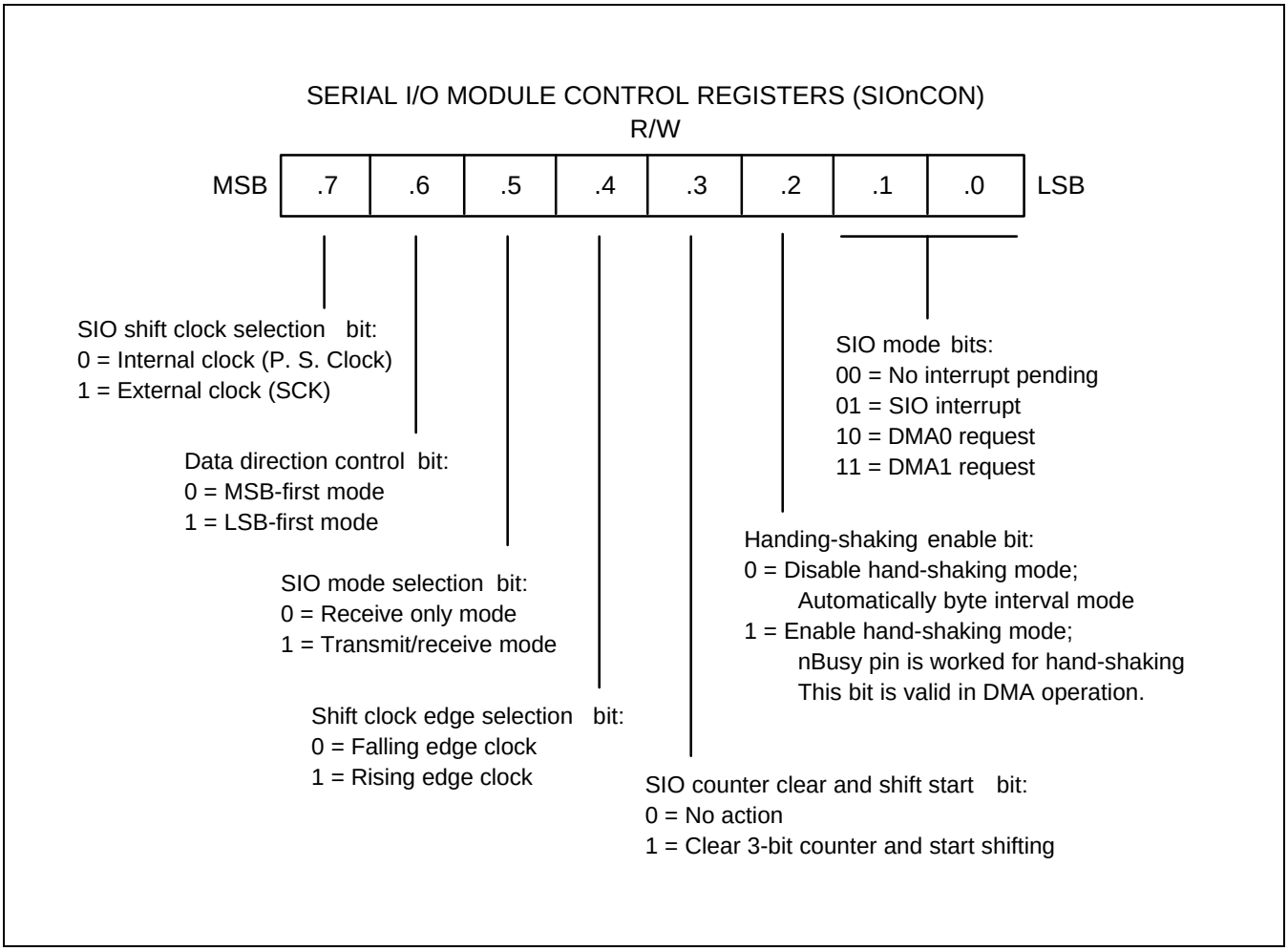


Figure 7-15. Serial I/O Module Control Registers (SIOnCON)

SIO 0/1 DATA REGISTER

The SIO0/1 data register, SIO0/1DATA, contains an 8-bit data value to be transmitted or received over the SIO 0/1 channel.

Register	Offset Address	R/W	Description	Reset Value
SIO0DATA	0x6002	R/W	Serial transmit/receive data 0 register	00h
SIO1DATA	0x7002	R/W	Serial transmit/receive data 1 register	00h

[7:0] Transfer data — This field contains the data to be transmitted or received over the SIO 0/1. When this register is written, the transmit data comes out through the SO0/1 pin by SCK0/1 synchronization.

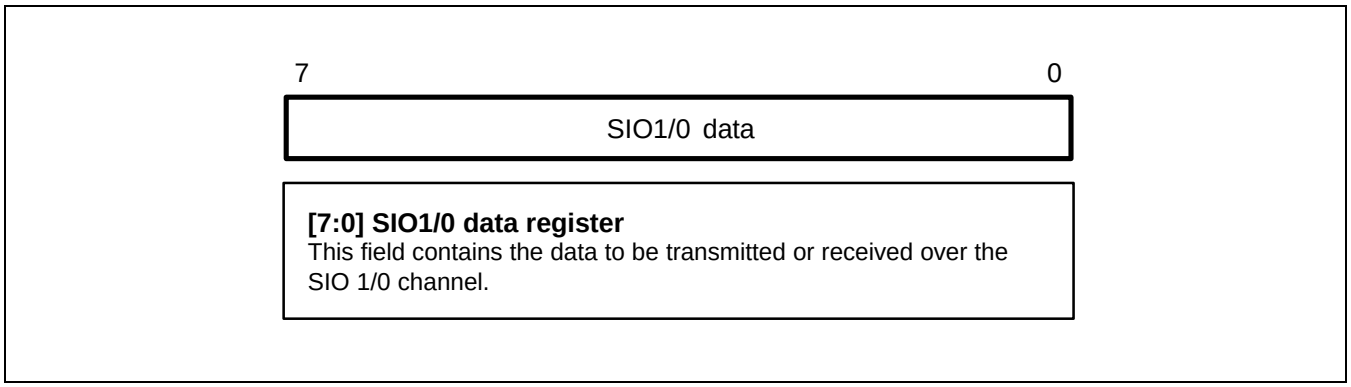


Figure 7-16. SIO0/1 Transmit/Receive Buffer Register (SIO0DATA, SIO1DATA)

SIO0/1 BAUD RATE PRESCALER REGISTERS

The value stored in the baud rate prescaler registers, SBRDR0/1, allows you to determine the SIO 0/1 clock rate (baud rate) as follows:

Baud rate = Baud input clock / 2(divisor value+1), where divisor > 0.

The baud input clock is either MCLK (the internal master clock) or SCK0/1 (the external SIO 0/1 clock input), as determined by the setting of the serial clock selection bit in the line control register, SIO0/1CON[6].

Register	Offset Address	R/W	Description	Reset Value
SBIDR0	0x6000	R/W	Byte interval data register for SIO0	00h
SBRDR0	0x6001	R/W	Baud rate divisor register for SIO0	00h
SBIDR1	0x7000	R/W	Byte interval data register for SIO1	00h
SBRDR1	0x7001	R/W	Baud rate divisor register for SIO1	00h

Sync SIO Baud rate = Source clock / 2 (8 bit Pre-scaler Value + 1), where the shift clock source is the baud rate clock.
Sync SIO Baud rate = SCKn, where the shift clock source is the SCKn clock input.

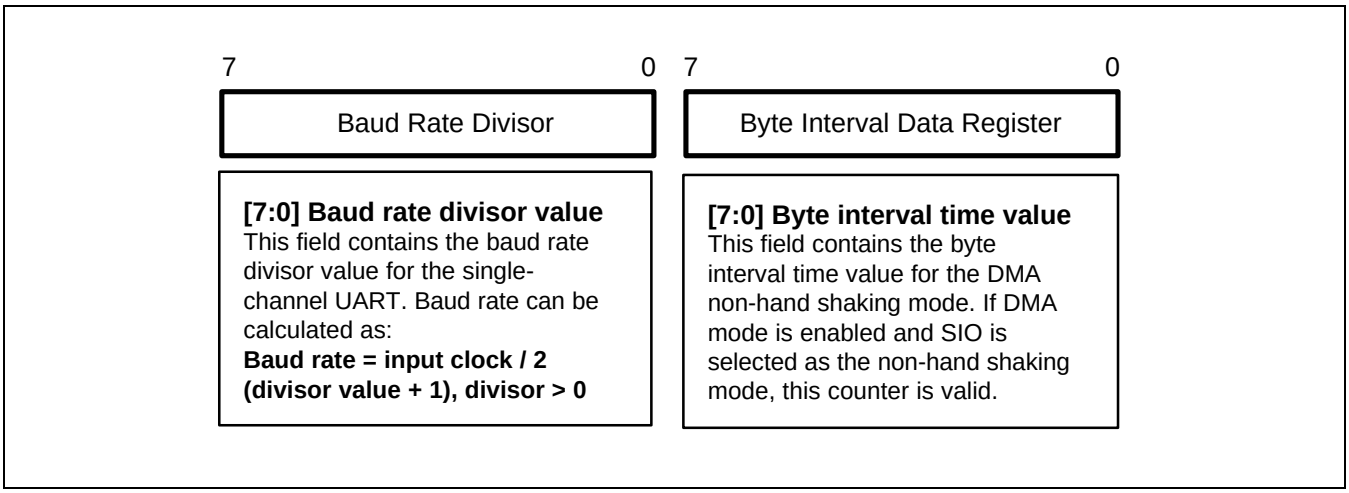


Figure 7-17. SIO 0/1 Baud Rate Divisor Registers (SBRDR) and Byte Interval Data Registers (SBIDR)

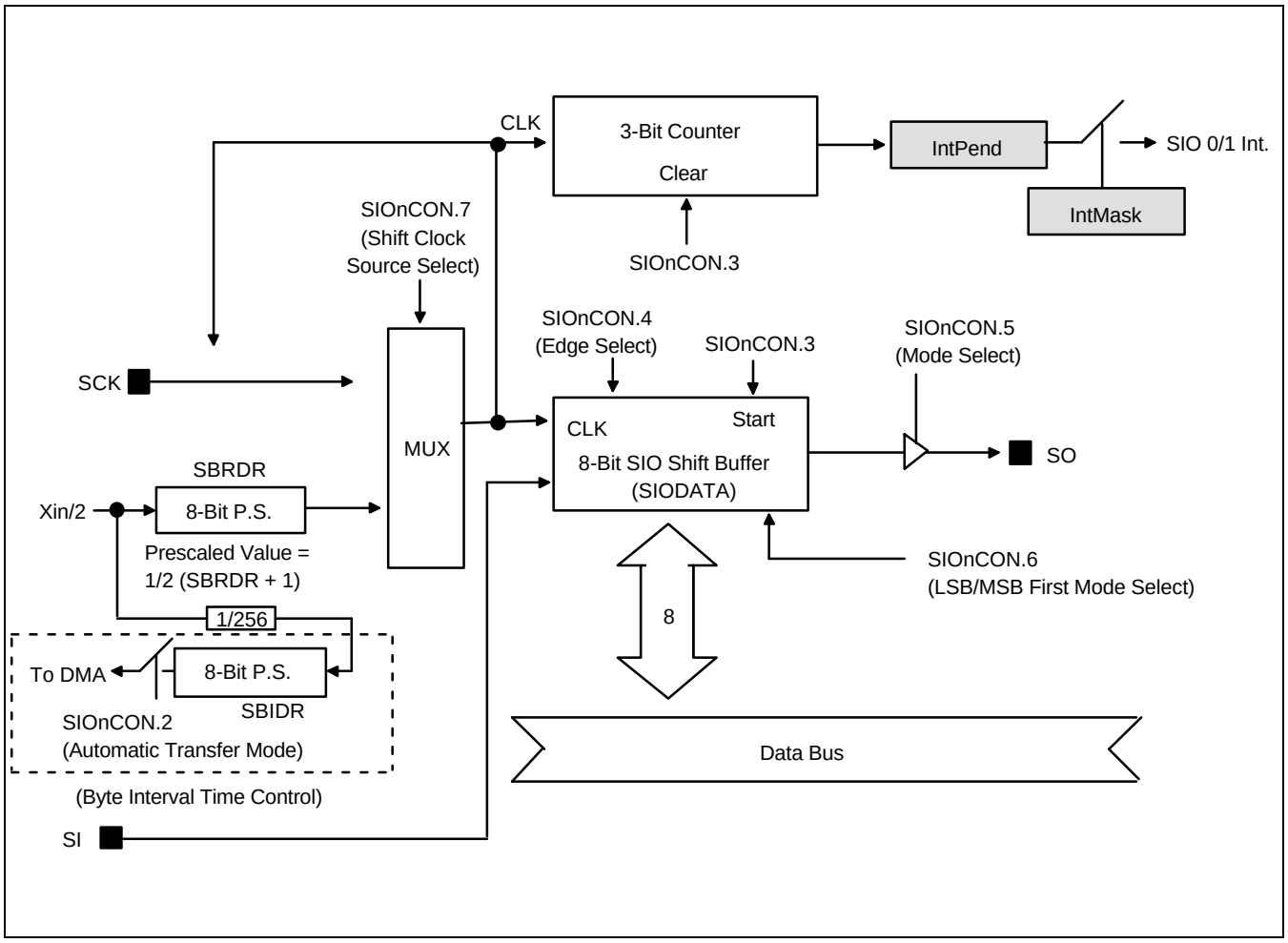


Figure 7-18. SIO Functional Block Diagram

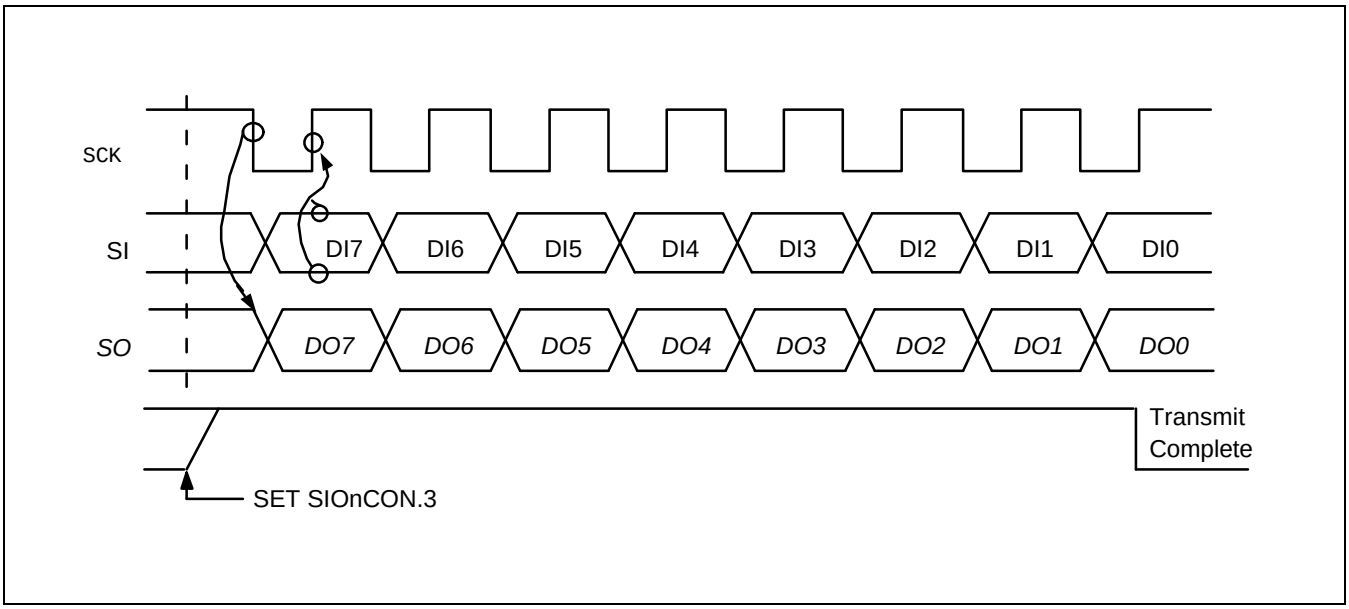


Figure 7-19. Serial I/O Timing Transmit/Receive Mode (Tx at Falling, SIOCON.4 = 1)

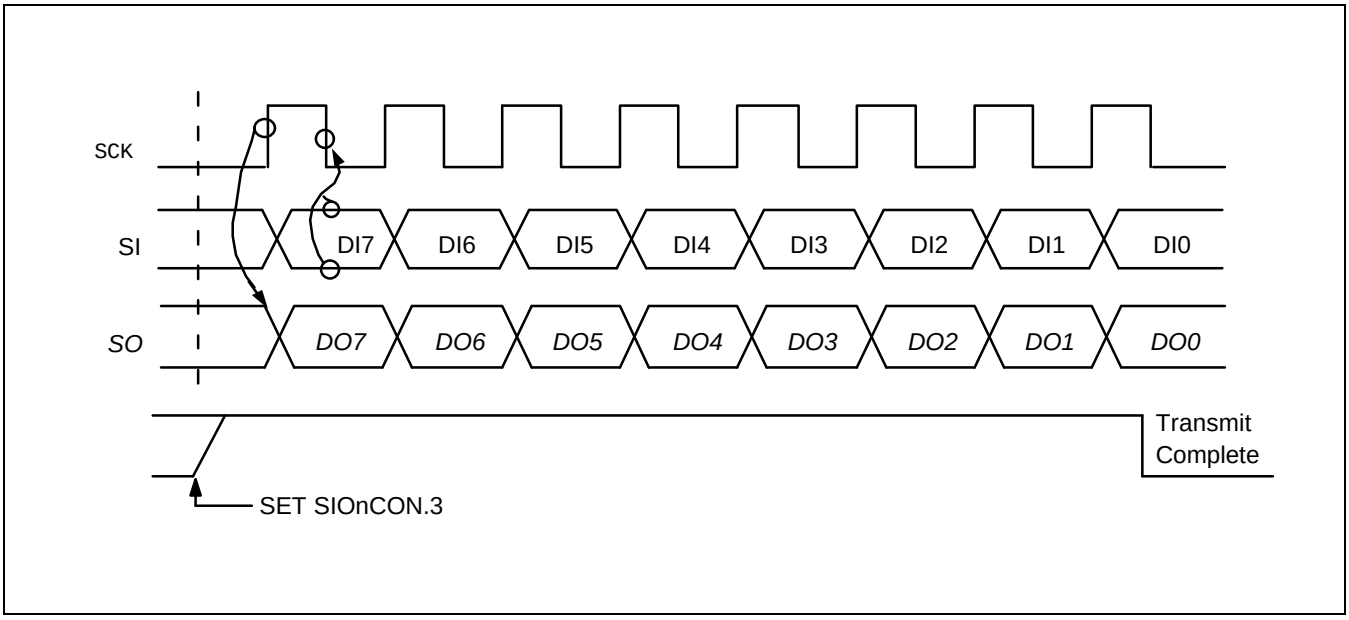


Figure 7-20. Serial I/O Timing Transmit/Receive Mode (Tx at Rising, SIOCON.4 = 0)

NOTES

8 BASIC TIMER & WATCHDOG TIMER

OVERVIEW

The KS17C4000 Basic Timer/Watchdog Timer is used to resume the controller operations when it is disturbed by noise or other kinds of system error or malfunctions. It can be used as a normal interval timer to request interrupt services. It also signals the end of the required oscillation interval after a reset or a Stop mode release. Users can control the disable or enable value for watchdog timer in BTCON.

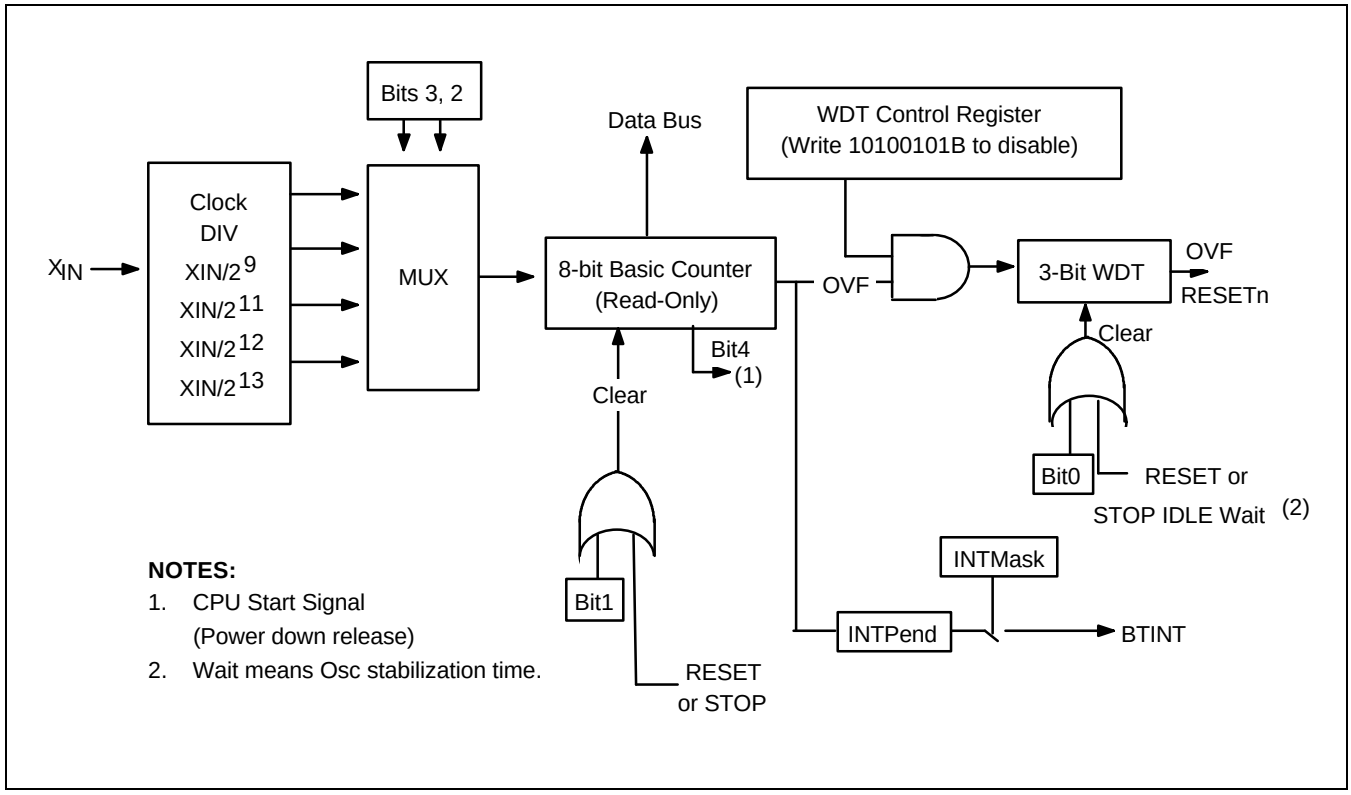


Figure 8-1. Basic Timer Block Diagram

BASIC TIMER COUNTER REGISTER

The basic timer counter register, BTCNT (Offset address: 0xa007), is used to specify the time out duration, and is a free run 8-bit counter.

Table 8-1. Basic Timer Counter Setting (at Xin = 20 MHz)

BTCON.3	BTCON.2	Clock source	Resolution	Interval Time	Max. interval	Remark
0	0	$X_{IN}/2^{13}$	409.6 us	$2^{13}/X_{IN} \times 2^8$	104.86 ms	Default setting
0	1	$X_{IN}/2^{12}$	204.8 us	$2^{12}/X_{IN} \times 2^8$	52.43 ms	
1	0	$X_{IN}/2^{11}$	102.4 us	$2^{11}/X_{IN} \times 2^8$	26.21 ms	
1	1	$X_{IN}/2^9$	25.6 us	$2^9/X_{IN} \times 2^8$	6.55 ms	

WATCHDOG TIMER COUNTER REGISTER

The watchdog timer counter register, WTCNT, is a free run 3-bit counter, used to specify the time out duration. The watchdog timer enable data can be any value, except 0xA5, to make the watchdog timer overflow, which will issue a system hardware reset.

Table 8-2. Watchdog Timer Counter Setting (at 20 MHz)

BTCON.3	BTCON.2	Clock source	Resolution	WDT interval.	Interval time	Remark
0	0	$X_{IN}/2^{13}$	409.6 us	$2^{13} / X_{IN} \times 2^8 \times 2^3$	838.86 ms	Default setting
0	1	$X_{IN}/2^{12}$	204.8 us	$2^{12} / X_{IN} \times 2^8 \times 2^3$	419.43 ms	
1	0	$X_{IN}/2^{11}$	102.4 us	$2^{11} / X_{IN} \times 2^8 \times 2^3$	209.72 ms	
1	1	$X_{IN}/2^9$	25.6 us	$2^9 / X_{IN} \times 2^8 \times 2^3$	52.43 ms	

Register	Offset Address	R/W	Description	Reset Value
BTCNT	0xa007	R	Basic Timer Count register	00h

BASIC TIMER CONTROL REGISTER

The basic timer control register, BTCON, contains watchdog counter enable bits, clock input setting bits, and counter clear bits.

Register	Offset Address	R/W	Description	Reset Value
BTCON	0xa002	R/W	Basic Timer Control register	0000h

Watchdog timer control register has the following bits:

[0]	WDT Counter clear bit	This bit clears the watchdog counter. When this bit is set to zero, the watchdog counter is automatically cleared.
[1]	Basic Counter clear bit	This bit clear the basic counter. When this bit is set to zero, the watchdog counter is automatically cleared.
[3:2]	Clock source select	These bits select a clock source. When it is 11, it selects a Xin/2^9 as a clock source. When 10, a Xin/2^{11}, when 01, a Xin/2^{12}, and, when 00, a Xin/2^{13}.
[15:8]	Watch dog timer enable	These bits control enabling or disabling the watchdog timer counting. When these bits are {10100101} value, watch dog timer counter stops. Other values can enable counting the watchdog timer, causing a system reset when an overflow signal occurs.

FUNCTION DESCRIPTION

INTERVAL TIMER FUNCTION

The basic timer's primary function is to measure elapsed time intervals. The standard time interval is equal to 256 basic timer clock pulses.

The 8-bit counter register, BTCNT, is incremented each time a clock signal is detected to be corresponding to the frequency selected by BTCON. BTCNT continues incrementing as it counts BT clocks until an overflow occurs (255). An overflow causes the BT interrupt pending flag to be set to logic one to signal that the designated time interval has elapsed. An interrupt request is then generated, BTCNT is cleared to logic zero, and the counting continues from 00H.

Oscillation Stabilization Interval Timer Function

You can also use the basic timer to program a specific oscillation stabilization interval after a reset or when Stop mode is released by an external interrupt.

In Stop mode, whenever a reset or an external interrupt occurs, the oscillator starts. The BTCNT value then starts increasing at the rate of f_{OSC}/8192 (for reset), or at the rate of the preset clock source (for an external interrupt). When BTCNT.4 is set, a signal is generated to indicate that the stabilization interval has elapsed and to gate the clock signal off to the CPU so that it can resume its normal operation.

In summary, the following events occur when Stop mode is released:

1. During Stop mode, a power-on reset or an external interrupt occurs to trigger a Stop mode release and oscillation starts.
2. If a power-on reset occurs, the basic timer counter will increase at the rate of f_{OSC}/8192. If an external interrupt is used to release Stop mode, the BTCNT value increases at the rate of the preset clock source.
3. Clock oscillation stabilization interval begins and continues until the bit 4 of the basic timer counter set.
4. When a BTCNT.4 set occurs, the normal CPU operation resumes.

WATCHDOG TIMER FUNCTION

The basic timer can also be used as a "watchdog" timer to detect inadvertent program loops, i.e., system or program operation errors. For this purpose, instructions that clear the watchdog timer within a given period should be executed at proper points in a program. If an instruction that clears the watchdog timer is not executed within the period and the watchdog timer overflows, a reset signal is generated and the system is restarted in the reset status. Operations of a watchdog timer are as follows:

— Each time BTCNT overflows, an overflow signal is sent to the watchdog timer counter, WDTCNT.

— If WDTCNT overflows, a system reset is generated.

A reset sets BTCON as #0000H. This value enables the watchdog timer. During normal operation, the application program must prevent the overflow. To do this, the WDTCNT value must be cleared (by writing a "1" to BTCON.0) at regular intervals.

9 TIMER MODULE 0 (16-BIT TIMERS)

OVERVIEW

The KS17C4000 has three 16-bit timers:T0,T1, and T2. These timers can operate in interval mode, capture mode, or match & overflow mode. The clock source for the timers can be an internal or an external clock. Or you can select one of these signals as the timer clock. You can enable or disable the timers by setting control bits in the corresponding timer mode register. Especially, the T0 can be also used as a time base of the real time output port (P0).

Timer 0,1, and 2 have three operating modes, one of which you select using the appropriate TnCON setting:

— Interval timer mode
— Capture input mode with a rising or falling edge trigger at the input pin(TnCAP)
— Match & Overflow mode

TIMER 0, 1, 2 CONTROL REGISTERS (T0CON, T1CON, T2CON)

You use the timer 0, 1, and 2 control registers, TnCON, to

— Select the timer n operating mode (interval timer, capture mode, or match & overflow mode)
— Select the timer n input clock (Internal or external clock)
— Clear the timer n counter, TnCNT

IntMask register controls to enabling or disabling the timer n overflow interrupt or timer n match/capture interrupt.

IntPend register contains timer n match/capture/overflow interrupt pending bits.

A reset clears TnCON to '00H'. This sets the timer n to normal interval timer mode, selects an input clock frequency of f_{PRE}, and disables all timer n interrupts. You can clear the timer n counter at any time during the normal operation by writing a "1" to TnCON.6.

INTERVAL MODE OPERATION

In interval timer mode, a match signal is generated when the counter value is identical to the value written to the Tn reference data register, TnDATA. The match signal generates a timer n match interrupt (TnINT) and clears the counter.

CAPTURE MODE OPERATION

In capture mode, the timer performs capture operation in which the counter value is fetched into the Buffer register in synchronization with an external trigger and retained there. A valid edge detected from the input of capture input pin is used as the external trigger. The counter value of 16-bit counter in the process of being counted is fetched into the capture register when an external trigger occurs. The contents of the capture register are retained until the next capture trigger is generated. If a trigger signal is not generated during the counter overflow, an overflow interrupt is generated and the counting continues from 0000H.

MATCH & OVERFLOW MODE OPERATION

In this mode, a match signal is generated when the counter value is identical to the value written to the timer n data register. However, the match signal does not clear the counter, while it can generate a match interrupt. Instead, it runs continuously, overflowing at FFFFH, and then continues incrementing from 0000H. When an overflow occurs, an overflow interrupt is generated. A new period begins and is repeated.

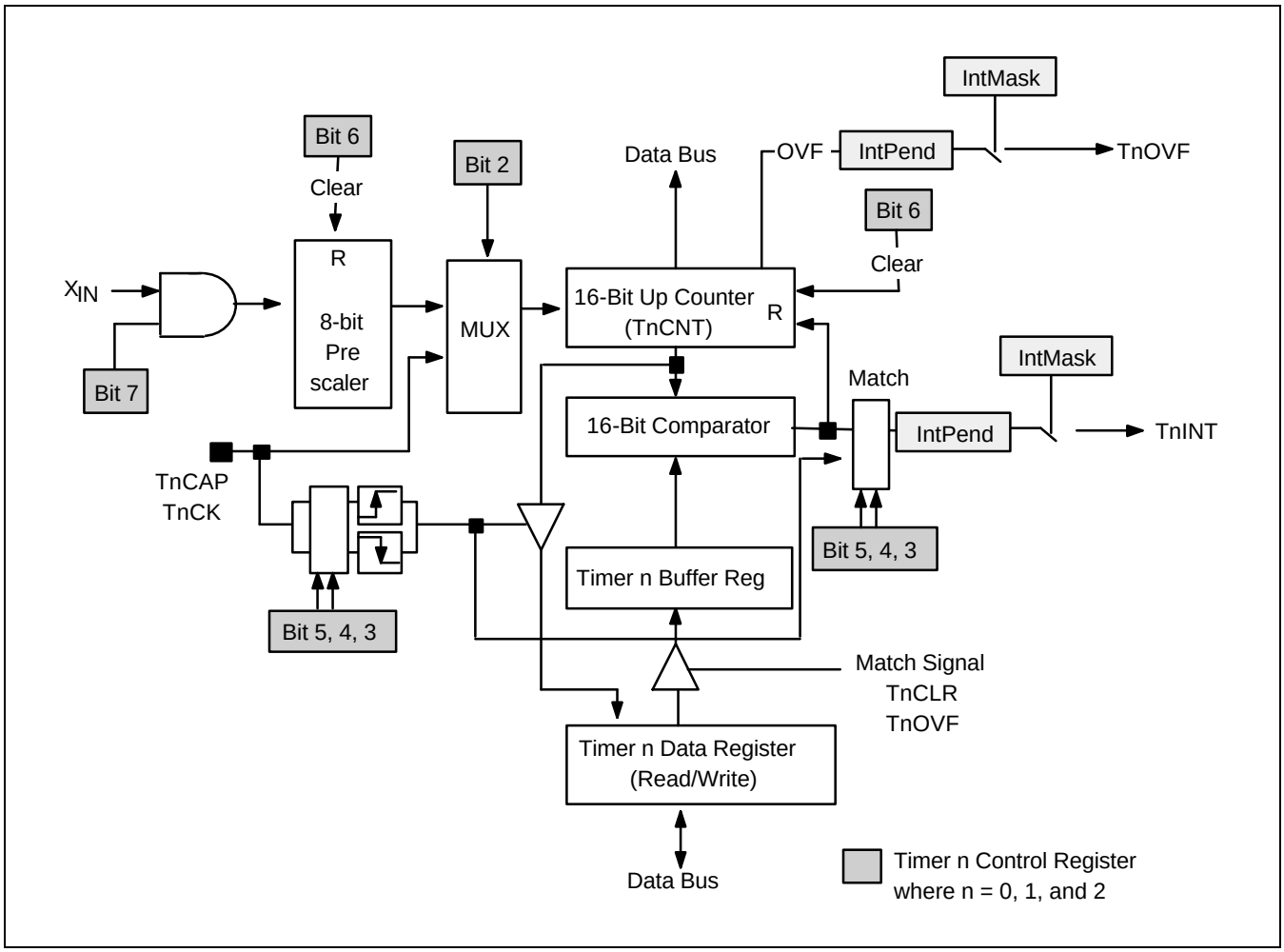


Figure 9-1. 16-Bit Timer Block Diagram

TIMER SPECIAL REGISTERS

TIMER CONTROL REGISTERS

The timer control registers, T0CON, T1CON, and T2CON, are used to control the operation of the three 16-bit timers.

Table 9-1. Timer Control Registers Description

Register	Offset Address	R/W	Description	Reset Value
T0CON	0x9003	R/W	Timer 0 control register	00h
T1CON	0x9013	R/W	Timer 1 control register	00h
T2CON	0x9023	R/W	Timer 2 control register	00h

Three timer mode registers have the following control settings:

[2]	Clock source selection	This field decides what clock source is used for the corresponding timer. When this bit is 0, the corresponding timer uses the internal pre-scaled clock (Fpre) as its clock source. When 1, an external clock source is selected.
[5:3]	Timer mode selection	This field determines what operation mode of the corresponding timer is used, interval, capture mode, or, match & overflow mode. When you set TnCON[5:3] to 000, the corresponding timer operates in interval mode. When a timer match signal occurs, a timer interrupt request can be generated. When 001, match & overflow mode. When you set TnCON[5:3] to 1xx, the corresponding timer operates in capture mode. When TnCON[5:3] is 100, the counter value is captured at falling edge on the capture input pin. When TnCON[5:3] is 101, the counter value is captured at rising edge on the capture input pin. When TnCON[5:3] is 110, the counter value is captured at falling and rising edge on the capture input pin.
[6]	Counter Clear bit	This bit controls clearing the counter register. When this bit is set the counter is cleared and this bit will be automatically cleared.
[7]	Timer enable/disable	You can enable or disable the timer by setting or clearing this bit. When TnCON[7] is 1, the corresponding timer is enabled and internal clock source mode is used.

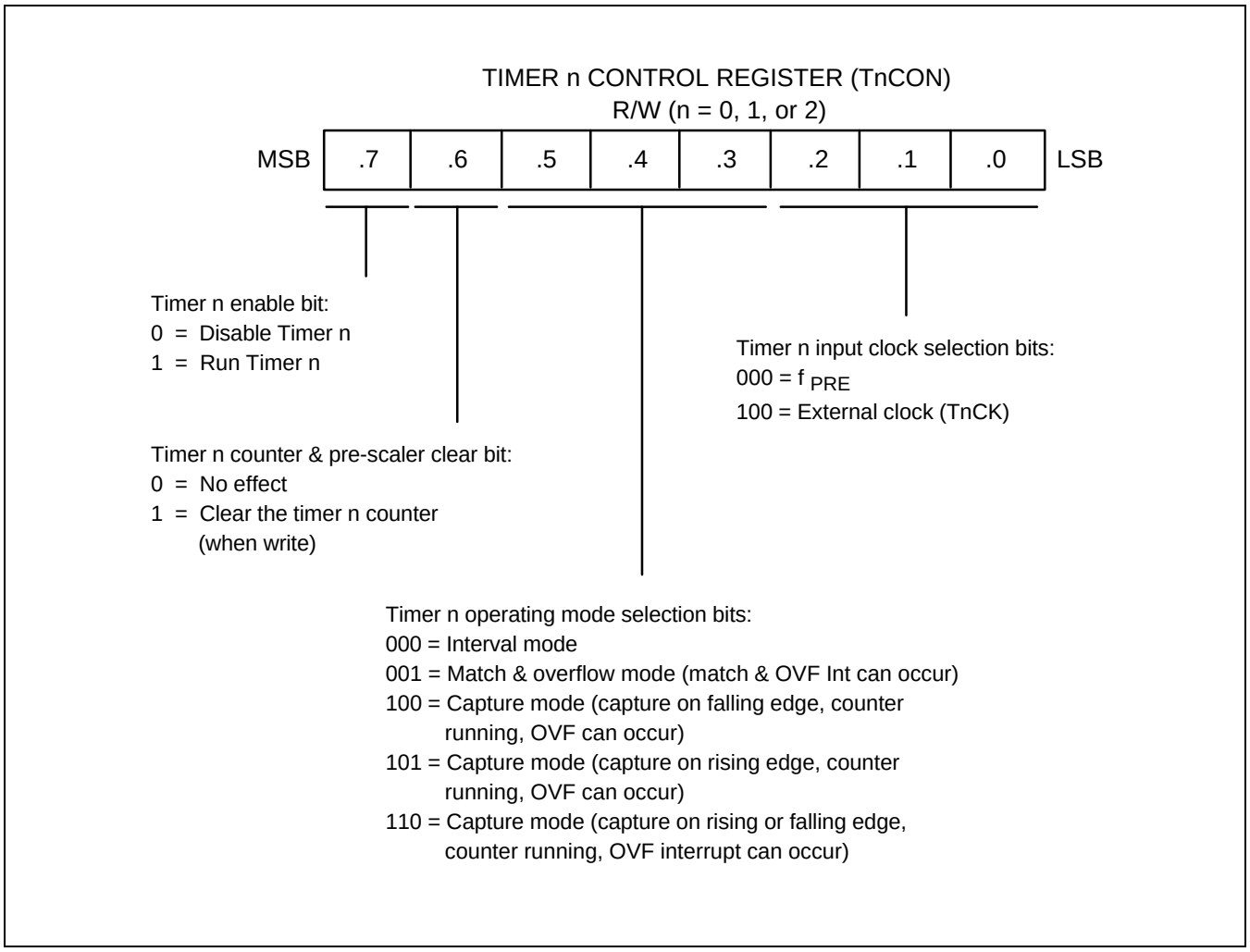


Figure 9-2. Timer 0, 1, 2 Control register

TIMER DATA REGISTERS

The timer data registers, T0DATA, T1DATA, and T2DATA, contain a value that specifies the time-out duration for each timer. The formula for calculating time-out duration is (Timer data + 1) cycles. See Figure 9-3 below.

Table 9-2. Timer Data Registers Description

Register	Offset Address	R/W	Description	Reset Value
T0DATA	0x9000	R/W	Timer 0 data register	ffffh
T1DATA	0x9010	R/W	Timer 1 data register	ffffh
T2DATA	0x9020	R/W	Timer 2 data register	ffffh

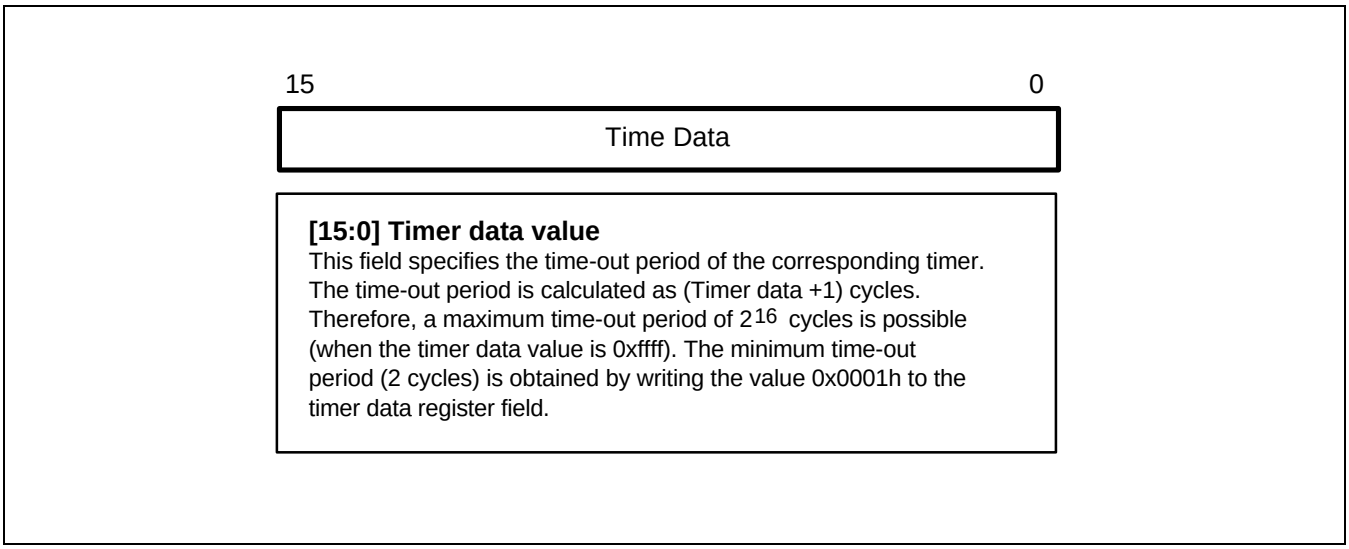


Figure 9-3. Timer Data Registers (T0DATA, T1DATA, and T2DATA)

TIMER COUNT REGISTERS

The timer count registers, T0CNT, T1CNT, and T2CNT, contain current timer 0, 1, and 2 count value, respectively, during the normal operation (see Figure 9-4).

Table 9-3 . Timer Count Registers description

Register	Offset Address	R/W	Description	Reset Value
T0CNT	0x9006	R	Timer 0 count register	0000h
T1CNT	0x9016	R	Timer 1 count register	0000h
T2CNT	0x9026	R	Timer 2 count register	0000h

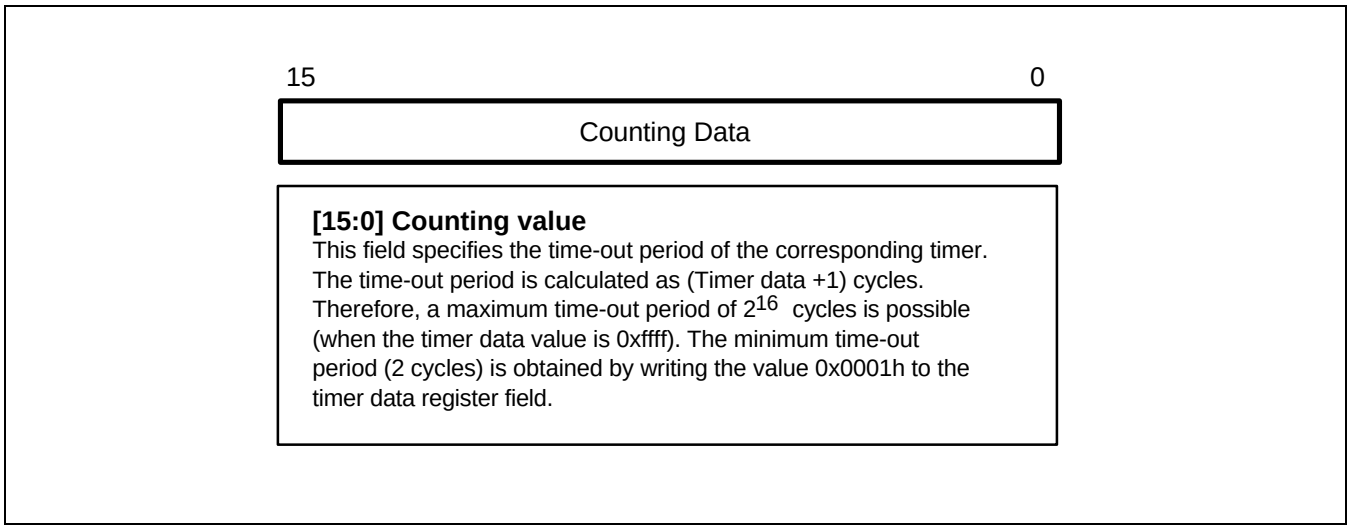


Figure 9-4. Timer Count Registers (T0CNT, T1CNT and, T2CNT)

TIMER PRESCALER REGISTERS

The timer pre-scaler registers, T0PRE, T1PRE, and T2PRE, contain a pre-counted value of the current timer 0, 1, and 2, respectively, during the normal operation (see Figure 9-5).

Table 9-4 . Timer Prescaler Registers

Register	Offset Address	R/W	Description	Reset Value
T0PRE	0x9002	R/W	Timer 0 prescaler register	ffh
T1PRE	0x9012	R/W	Timer 1 prescaler register	ffh
T2PRE	0x9022	R/W	Timer 2 prescaler register	ffh

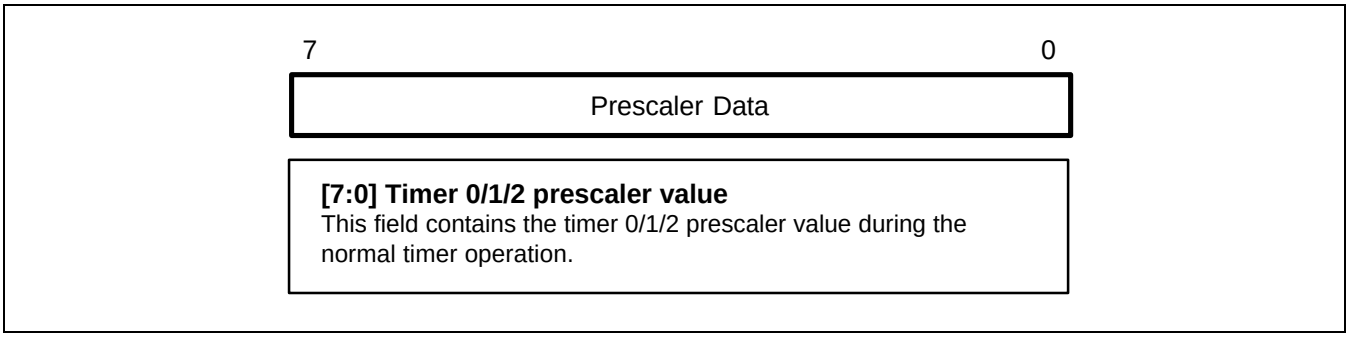


Figure 9-5. Timer Prescaler Registers (T0PRE, T1PRE, and T2PRE)

Prescaler register contains an 8-bit prescaler value. If the prescaler value is a n, the prescaler factor is calculated as n+1.

NOTES

10 TIMER MODULE 1 (8-BIT TIMERS)

OVERVIEW

The KS17C4000 has two 8-bit timers. These timers can operate in interval mode, capture mode or PWM (Pulse Width Modulation) mode. The clock source for the timers can be an internal or an external clock. Or you can select one of these signals as the timer clock. You can enable or disable the timers by setting control bits in the corresponding timer mode register. Especially, T3 can also be used as a time base of the real time output port (P0).

Timer 3/4 has three operating modes, one of which you select using the appropriate T3CON or T4CON setting:

— Interval timer mode (Toggle output at TnOUT pin)

— Capture input mode with a rising or falling edge trigger at the TnCAP pin

— PWM mode (TnPWM)

TIMER 3, 4 CONTROL REGISTERS (T3CON, T4CON)

You can use the timer 3/4 control register, T3/4CON, to

— Select the timer 3/4 operating mode (interval timer, capture mode, or PWM mode)

— Select the timer 3/4 input clock frequency

— Clear the timer 3/4 counter, TnCNT

IntMask register controls enabling or disabling the timer 3/4 overflow interrupt or timer 3/4 match/capture interrupt. IntPend register contains timer 3/4 match/capture/overflow interrupt pending bits.

A reset clears T3CON and T4CON to '00H'. This sets timer 3/4 to normal interval timer mode, selects an input clock frequency of $f_{OSC}/(256 \times 16)$, and disables all timer 3/4 interrupts. You can clear the timer 3/4 counter at any time during the normal operation by writing a "1" to TnCON.6.

INTERVAL MODE OPERATION

In interval timer mode, a match signal is generated and TnOUT is toggled when the counter value is identical to the value written to the Tn reference data register, TnDATA. The match signal generates a timer n match interrupt (TnINT) and clears the counter.

If, for example, you write the value 10H to TnDATA and C0H to TnCON, the counter will increment until it reaches to 10H. At this point, an interrupt request is generated, the counter value is reset, and counting resumes.

CAPTURE MODE OPERATION

In capture mode, the timer performs capture operation in which the counter value is fetched into the Buffer register in synchronization with an external trigger and retained. A valid edge detected from the input of capture input pin is used as the external trigger. The counter value of 8-bit counter in the process of being counted is fetched into the capture register when external trigger occurs. The contents of the capture register are retained until the next capture trigger is generated. If no trigger signal is not generated during counter overflow, Overflow interrupt can be generated and continues to count from 00H.

PWM MODE OPERATION

Pulse width modulation (PWM) mode allows you to program the width (duration) of the pulse that is output at the TnPWM pin. The PWM output which has the programmable duty and fixed frequency is coming out at the configured output pin.

In this mode, the match signal is generated when the counter value is identical to the value written to the timer n data register. However, the match signal does not clear the counter, while it can generate a match interrupt. Instead, it runs continuously, overflowing at FFH, and then continues incrementing from 00H. When an overflow occurs, an overflow interrupt is generated. A new period begin and the operation is repeated.

Although you can use the match signal to generate a timer overflow interrupt, interrupts are not typically used in PWM-type applications. Instead, the pulse at the TnPWM pin is held to Low level as long as the reference data value is *less than or equal to* (≤) the counter value and then the pulse is held to High level for as long as the data value is *greater than* (>) the counter value. One pulse width is equal to $t_{CLK} \times 256$.

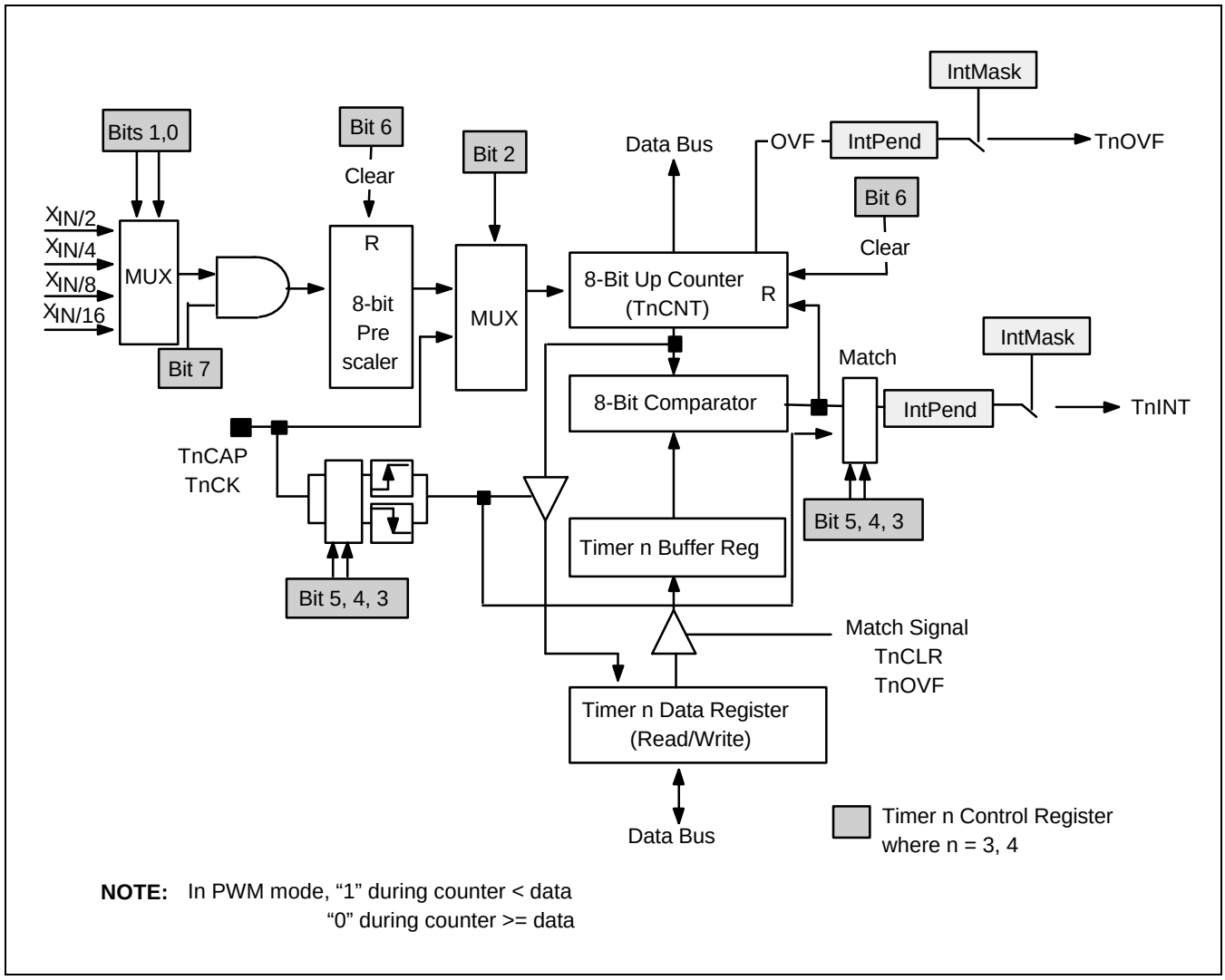


Figure 10-1. 8-Bit Timer 3/4 Block Diagram

TIMER SPECIAL REGISTERS

TIMER CONTROL REGISTERS

The timer control registers, T3CON and T4CON, are used to control the operation of the two 8-bit timers.

Register	Offset Address	R/W	Description	Reset Value
T3CON	0x9033	R/W	Timer 3 control register	00h
T4CON	0x9043	R/W	Timer 4 control register	00h

Both timer mode registers have the following control settings:

[2:0]	Clock source selection	This field decides what clock source is used for the corresponding timer. When this bit is 000, the corresponding timer uses the internal pre-scaled clock (Fpre/16) as its clock source. When TnCON[1:0] is 001, the internal pre-scaled clock (Fpre/8) as its clock source, when 010, the internal pre-scaled clock (Fpre/4), when 011, the internal pre-scaled clock (Fpre/2), and, when 100, an external clock source is selected.
[5:3]	Timer mode selection	This field determines what operation mode of the corresponding timer is used for interval, capture, and PWM mode. When you set TnCON[5:3] to 000, the corresponding timer operates in interval mode. When a timer match signal occurs, timer interrupt request can be generated TnOUT is toggled. When you set TnCON[5:3] to 001, the corresponding timer operates in PWM mode. In PWM mode, the external TnPWM pin provide the pulse with variable duty and fixed frequency. The duty of PWM may have 256kinds from 0 to 255 step according to TnDATA register value. The PWM output come out low state till match signal occurs and then come out high state before next period begins. In PWM mode, match interrupt or overflow interrupt can be used. When you set TnCON[5:3] to 1xx, the corresponding timer operates in capture mode. When TnCON[5:3] is 100, counter value is captured at falling edge on capture input pin. When TnCON[5:3] is 101, counter value is captured at rising edge on capture input pin. When TnCON[5:3] is 110, counter value is captured at falling and rising edge on capture input pin.
[6]	Counter Clear bit	This bit controls to clear the counter register. When this bit is set the counter is cleared and this bit automatically will be cleared.
[7]	Timer enable/disable	You can enable or disable the timer by setting or clearing this bit. When TnCON[7] is 1, the corresponding timer is enabled when internal clock source mode is selected.

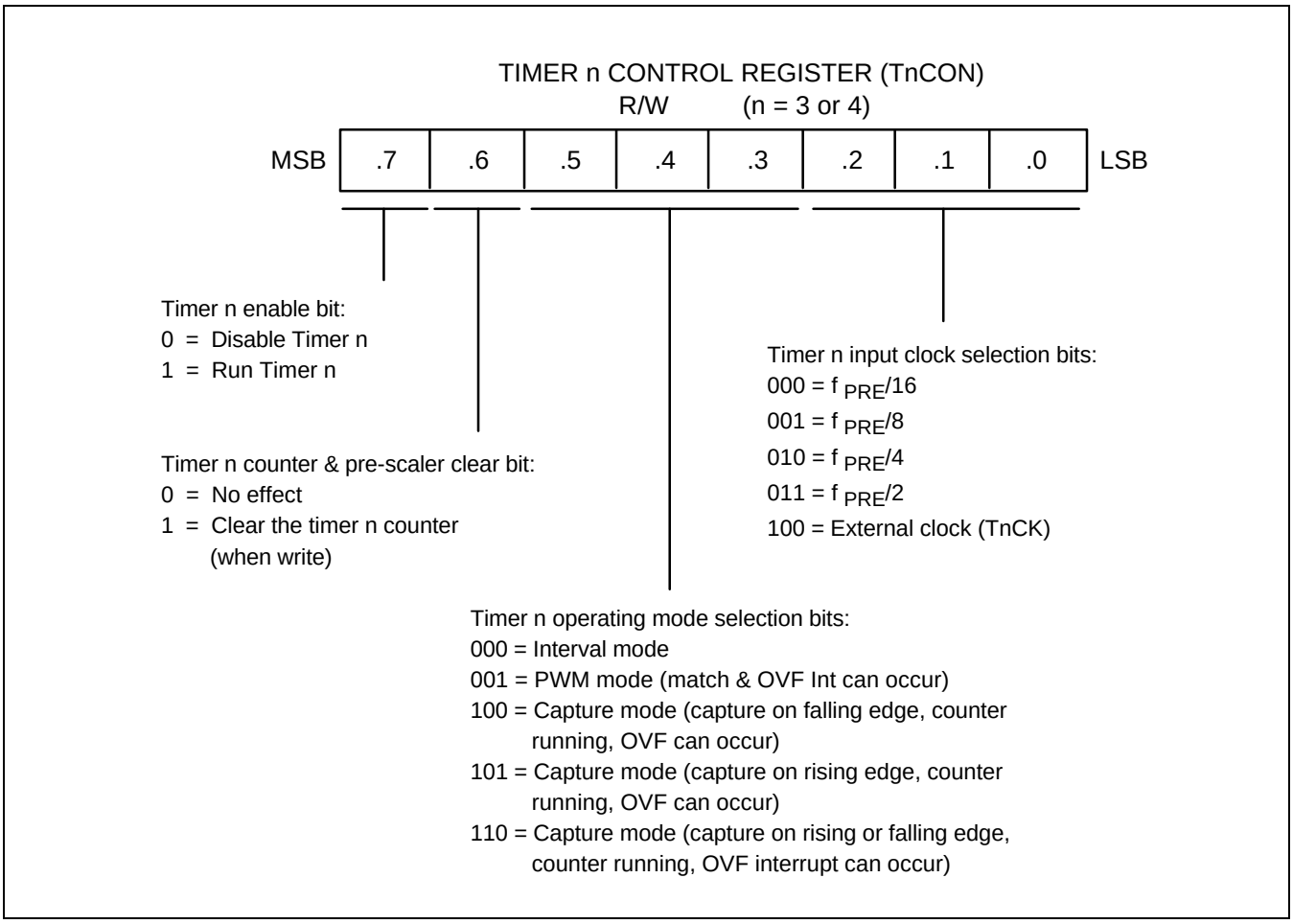


Figure 10-2. Timer n Control Registers (T3CON/T4CON)

TIMER DATA REGISTERS

The timer data registers, T3DATA and T4DATA, contain a value that specifies the time-out duration for each timer. The formula for calculating time-out duration is (Timer data + 1) cycles. See Figure 10-3 below.

Register	Offset Address	R/W	Description	Reset Value
T3 DATA	0x9031	R/W	Timer 3 data register	ffh
T4 DATA	0x9041	R/W	Timer 4 data register	ffh

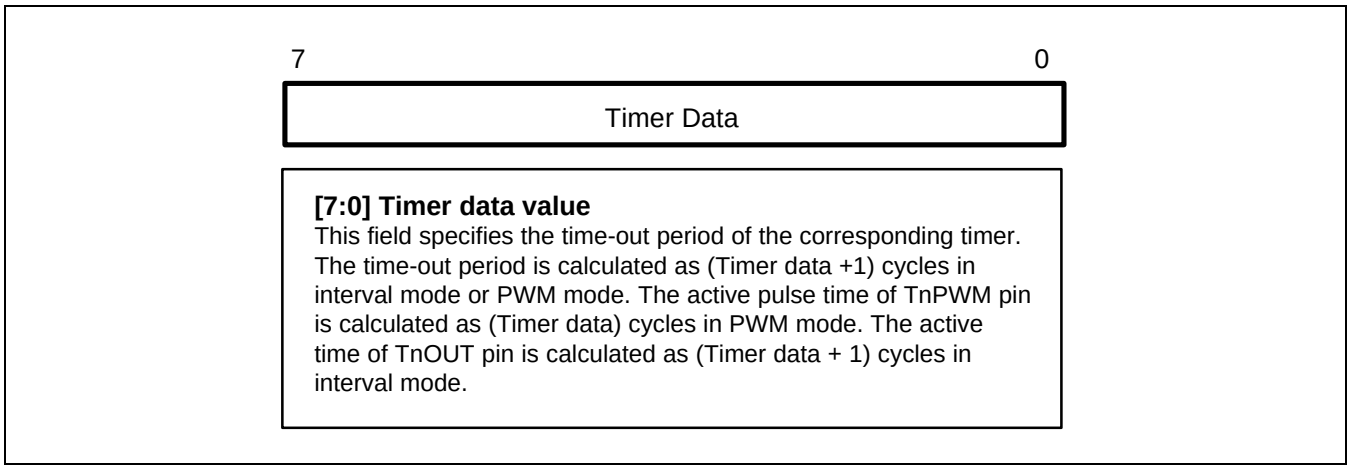


Figure 10-3. Timer Data Registers (T3DATA , T4DATA)

TIMER PRESCALER REGISTERS

The timer prescaler registers, T3PRE and T4PRE, contain a pre-counted value to the current timer 3 and 4, respectively, during the normal operation (see Figure 10-4).

Register	Offset Address	R/W	Description	Reset Value
T3PRE	0x9032	R/W	Timer 3 prescaler register	ffh
T4PRE	0x9042	R/W	Timer 4 prescaler register	ffh

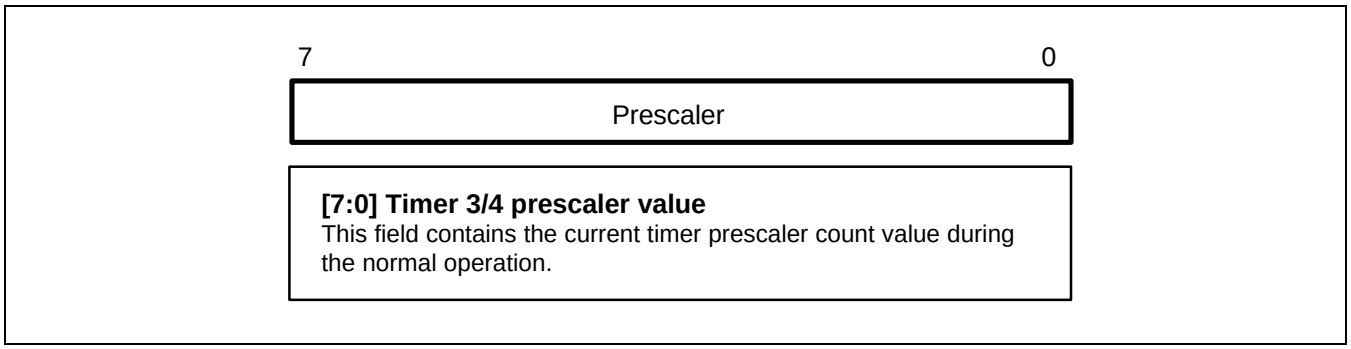


Figure 10-4. Timer Prescaler Registers (T3CNT, T4CNT)

Prescaler register contains an 8-bit prescaler value. If prescaler value is n, the prescaler factor is calculated as n+1.

TIMER COUNT REGISTERS

The timer count registers, T3CNT and T4CNT, contain a current timer 3 and timer 4 count value, respectively, during the normal operation (see Figure 10-5).

Register	Offset Address	R/W	Description	Reset Value
T3CNT	0x9037	R	Timer 3 count register	00h
T4CNT	0x9047	R	Timer 4 count register	00h

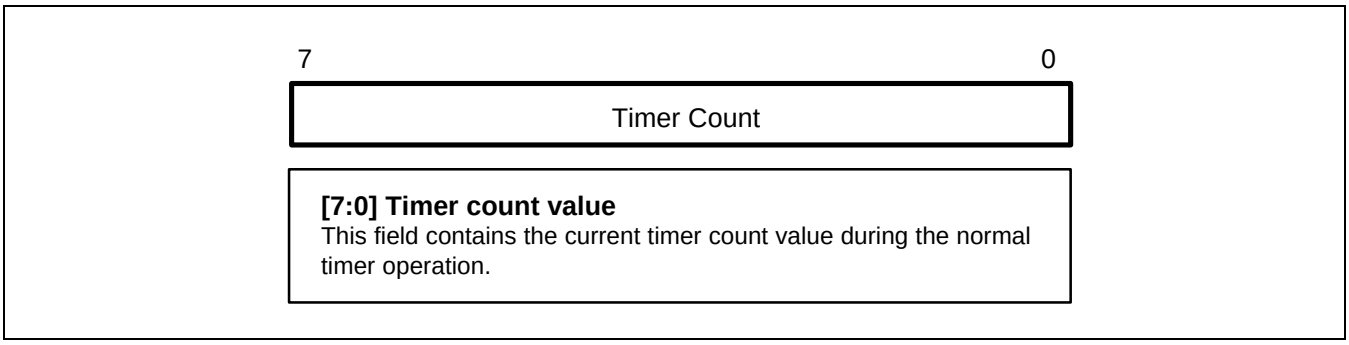


Figure 10-5. Timer Count Registers (T3CNT, T4CNT)

NOTES

11 8-BIT ANALOG-TO-DIGITAL CONVERTER

OVERVIEW

The KS17C4000 has eight 8-bit resolution A/D converter input pins(ADC0 to ADC7). The 8-bit A/D converter (ADC) module uses successive approximation logic to convert analog levels entering at one of the eight input channels to equivalent 8-bit digital values. The analog input level must lie between the AV_{REF} and AV_{SS} values. The A/D converter has the following components:

— Analog comparator with successive approximation logic
— D/A converter logic (resistor string type)
— ADC control register (ADCON)
— Eight multiplexed analog data input pins (ADC0-ADC7)
— 8-bit A/D conversion data output register (ADDATA)
— 8-bit digital input port
— AV_{REF} and AV_{SS} pins

FUNCTION DESCRIPTION

To initiate an analog-to-digital conversion procedure, you should write the channel selection data in the A/D converter control register, ADCON, to select one of the eight analog input pins (ADCn, n = 0-7) and set the conversion start or enable bit, ADCON.0. The conversion loads data to ADDATA register, and an INTAD interrupt can be generated.

During a normal conversion, A/D C logic initially sets the successive approximation register to 80H (the approximately the half-way point of an 8-bit register). This register is then updated automatically during each conversion step. The successive approximation block performs 8-bit conversions for one input channel at a time. You can dynamically select different channels by manipulating the channel selection bit value (ADCON.6-4) in the ADCON register. To start the A/D conversion, you should set the enable bit, ADCON.0. When a conversion is completed, ADCON.3, the end-of-conversion (EOC) bit, is automatically set to 1 and the result is dumped into the ADDATA register where it can be read. The A/D converter then enters the idle state. Remember to read the contents of ADDATA before another conversion starts, otherwise, the previous result will be overwritten by the next conversion result.

NOTE

Because the A/D converter has no sample-and-hold circuitry, it is very important that the fluctuation in the analog level at the ADC0-ADC7 input pins during a conversion procedure be kept to an absolute minimum. Any change in the input level, perhaps due to noise, will invalidate the result. If the chip enters STOP or IDLE mode in the conversion process, there must be a leakage current path in the A/D block. You must use STOP or IDLE mode after the A/D C operation is finished.

CONVERSION TIMING

The A/D conversion process requires 4 steps (4, 16, 32, or 64 clock edges) to convert each bit and 2 steps to set up the A/D Converter block. Therefore, a total of 36 steps are required to complete an 8-bit conversion. One step can be 1, 4, 8, or 16 clocks depending on the software.

In a 20 MHz CPU clock frequency, one clock cycle is 50 ns. The conversion rate is calculated as follows:

4 steps/bit × 8 bits + set-up time (2 steps time) = 34 Steps (1.7us, 6.8us, 13.6us, or 27.2us at 20 MHz)

To get the correct A/D Conversion result data, A/D Conversion time should be longer than 8 us whatever the oscillation frequency is used.

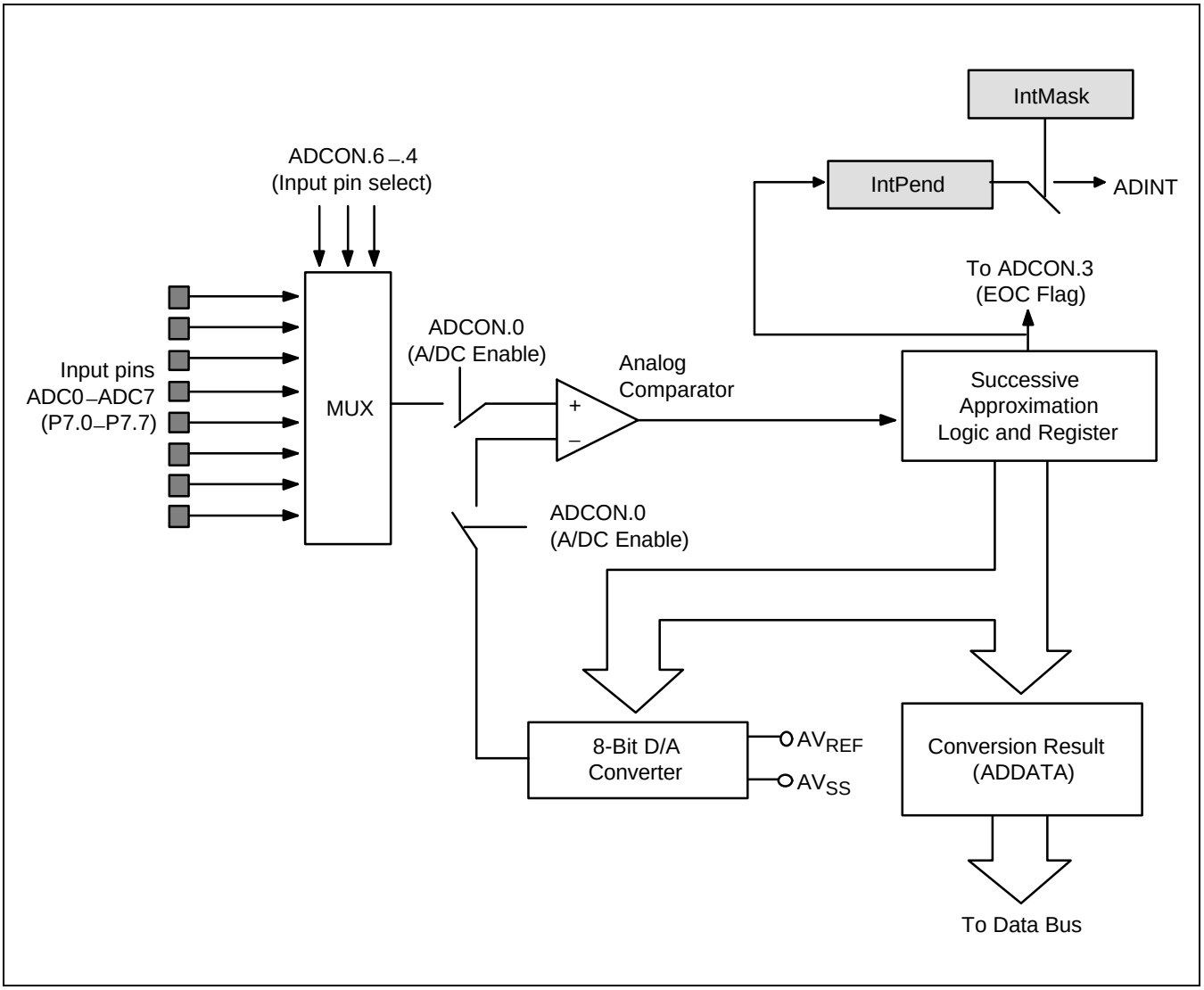


Figure 11-1. A/D C Block Diagram

A/D C SPECIAL REGISTERS

A/D C CONTROL REGISTERS

The A/D C control registers, ADCON, is used to control the operation of the eight 8-bit A/D C channels.

Register	Offset Address	R/W	Description	Reset Value
ADCON	0x8003	R/W	A/D C control register	08h

A/D Converter control register has the following control bit settings:

[0]	ADSTR	0: A/D conversion is disabled. 1: A/D conversion begins and is cleared after conversion.
[2:1]	Select the Conversion Speed	00: Step clock = Fx / 16 01: Step clock = Fx / 8 10: Step clock = Fx / 4 11: Step clock = Fx / 1
[3]	EOC (read-only)	0: Conversion is not completed. 1: This flag is set after conversion.
[6:4]	A/D C input select	000: select a ADC0 001: select a ADC1 010: select a ADC2 011: select a ADC3 100: select a ADC4 101: select a ADC5 110: select a ADC6 111: select a ADC7
[7]	Not used.	

A/D CONVERTER DATA REGISTERS

The A/D Conversion data register, ADDATA, contains the conversion result value that specifies the analog input channel.

Register	Offset Address	R/W	Description	Reset Value
ADDATA	0x8007	R	A/DC Conversion Result data register	xxh

A/D Converter data register has the following bits:

[0:7]	A/D C Data	This register has an A/D conversion result value.

NOTES

12 I/O PORTS

OVERVIEW

Of the 100 pins in the KS17C4000's QFP package, 62 pins are used for I/O. There are eight ports:

— Six 8-bit I/O ports (port 0, 1, 2, 3, 5, and 6)

— One 6-bit I/O port (port 4)

— One 8-bit input port (ADC0-ADC7/P7.0-P7.7)

Each port can be easily configured by software to meet various system configuration and design requirements. As the CPU accesses I/O ports by directly writing or reading port register addresses, no special I/O instruction is needed.

You can configure port 2 as address and data bus signal lines, and port 3 as control signal lines for the device's external interface. Port 7 can be used as analog inputs for the A/D converter module, or as general input port pins.

Table 12-1. KS17C4000 Port Configuration Overview

Port	Configuration Options	Programmability
0	General C-MOS push-pull I/O port with pull-up resistor assigned by software control. P0 can be used alternately as a real time output port.	Bit programmable
1	General I/O port with pull-down resistor; can alternately serve as external interrupt inputs, INT2-INT9, or can be configured as address lines, A16-A23 for external interface.	Bit programmable
Data	External interface data line (lower); can be configured as data lines, D0-D7 for external interface.	Fixed
Address	External interface address line (lower); can be configured as address lines A0-A15, or multiplexed A8/A16-A15/A23 for external interface.	Fixed
2	General I/O port with pull-down resistor; can be configured as address lines (higher), A16-A23, or data lines, D8-D15, for external interface.	Bit programmable
3	General I/O port with pull-up or down resistor; P3.0, P3.1, P3.2, P3.3, P3.4, P3.5, P3.6, and P3.7 can be used alternately as bus control signal lines for external interface; nWBE1, nCS0, nRAS1/nCS1, nRAS2/nCS2, nRAS3/nCS3, nCAS0/nECS0, nCAS1/nECS1, and nSWE/nAS.	Bit programmable
4	General I/O port; P4.0 can be used as device nWAIT/nBusy signal input. P4.1/P4.3 can be used alternately as an external request input for DMA module; nXDREQ. P4.2/P4.4 can be used alternately as external acknowledge output for DMA request; nXDACK. P4.5 can be used as an external interrupt input INT1;	Bit programmable
5	General I/O port assignable pull-up resistor; P5 can be used alternately as inputs or outputs for UART, SIO0, or SIO1 module.	Bit programmable
6	General I/O port; P6.0, P6.1, and P6.2 can alternately serve as an external capture input or clock input for Timer 0, Timer 1, and Timer 2, respectively. P6.3 and P6.4 can alternately serve as external clock input for Timer 3 and Timer 4, respectively. P6.5, and P6.6 can alternately serve as PWM/Toggle Out output or capture input for Timer 3 and Timer 4, respectively. P6.7 can be used as an external interrupt inputs INT0;	Bit programmable
7	Analog input channels, ADC0-ADC7; alternately, general input port or external interrupt input, INT10, INT11, and INT12	Bit programmable

PORT DATA REGISTERS

All eight port data registers have the identical structure as shown in Figure 12-1 below:

Table 12-2. Port Data Register Summary

Register Name	Mnemonic	Hex	Reset Value	R/W
Port 0 Data Register	P0	b000	00h	R/W
Port 1 Data Register	P1	b001	00h	R/W
Port 2 Data Register	P2	b002	00h	R/W
Port 3 Data Register	P3	b003	00h	R/W
Port 4 Data Register	P4	b004	00h	R/W
Port 5 Data Register	P5	b005	00h	R/W
Port 6 Data Register	P6	b006	00h	R/W
Port 7 Data Register	P7	b007	xxh	R
Port 0 Buffer Register	P0BR	b008	00h	R/W

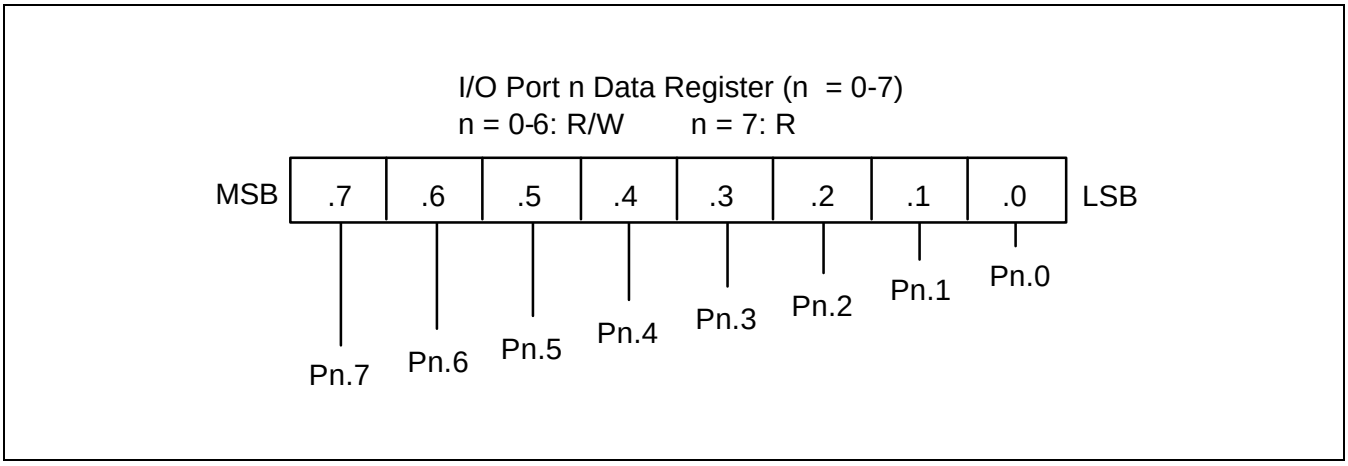


Figure 12-1. Port Data Register Structure

Table 12-3. Port Control Register Summary

Register Name	Mnemonic	Hex	Reset Value	R/W
External Interrupt Control Register	EINTCON	b00a	0000h	R/W
External Interrupt Mode Register	EINTMOD	b00c	000 0000h	R/W
Port 0 Control Register	P0CON	b010	000h	R/W
Port 1 Control Register	P1CON	b012	0000h	R/W
Port 2 Control Register	P2CON	b014	0000h	R/W
Port 3 Control Register	P3CON	b018	0000h	R/W
Port 4 Control Register	P4CON	b01a	000h	R/W
Port 5 Control Register	P5CON	b01c	00 0000h	R/W
Port 6 Control Register	P6CON	b020	000h	R/W
Port 7 Control Register	P7CON	b022	00h	R/W
P0 Pull-up Register	P0PUR	b028	00h	R/W
P1 Pull-down Register	P1PDR	b029	ffh	R/W
P2 Pull-down Register	P2PDR	b02a	ffh	R/W
P3 Pull-up/down Register	P3PUR	b02b	ffh	R/W
P4 Pull-up Register	P4PUR	b02c	0h	R/W
P5 Pull-up Register	P5PUR	b02d	00h	R/W
P6 Pull-up Register	P6PUR	b02e	00h	R/W
P7 Pull-up Register	P7PUR	b02f	00h	R/W

PORT 0

Port 0 circuit is shown in Figure 12-2 below:

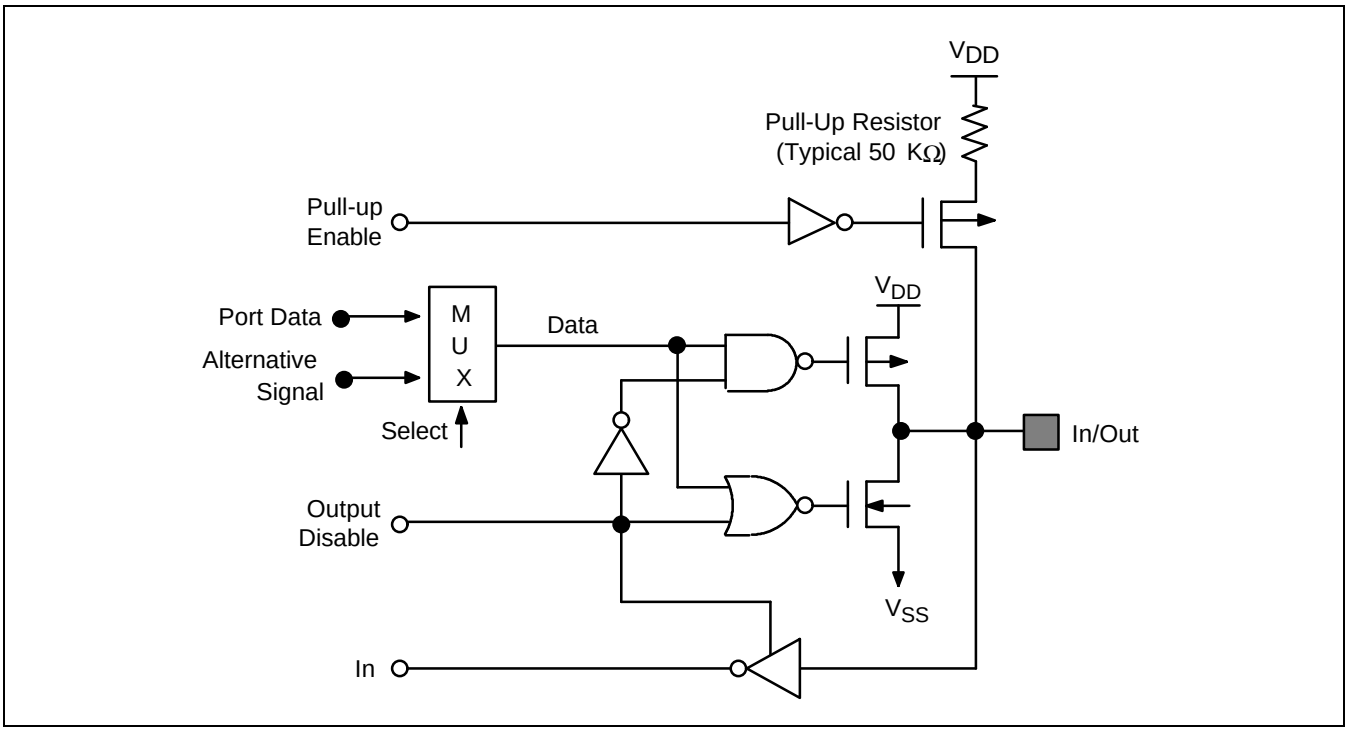


Figure 12-2. Pin Circuit Type 0 (Port 0)

Table 12-4. Port 0 Control Register

Name	Bit	Setting	Description
P0CON	0, 1, 2, 3, 4, 5, 6, 7	0 or 1	Setting the corresponding bit of Port 0. 0: CMOS input mode 1: CMOS push-pull output mode
	9, 8	0 or 1	Setting Port 0 as the real time output. 00: Normal in/output mode 01: Low nibble real time output buffer (P0BR.0-.3) mode 10: High nibble real time output buffer (P0BR.4-.7) mode 11: 8-bit real time output buffer (P0BR.0-.7) mode
	10	0 or 1	Time source of low nibble real time output 0: T0, 1: T3
	11	0 or 1	Time source of high nibble real time output 0: T0, 1: T3
P0PUR	0, 1, 2, 3, 4, 5, 6, 7	0 or 1	Setting the corresponding pull-up resistor of Port 0. 0: Disable pull-up resistor 1: Enable pull-up resistor
P0BR	0, 1, 2, 3, 4, 5, 6, 7	0 or 1	Setting the corresponding buffer of Port 0. 0: Data low, 1: Data high

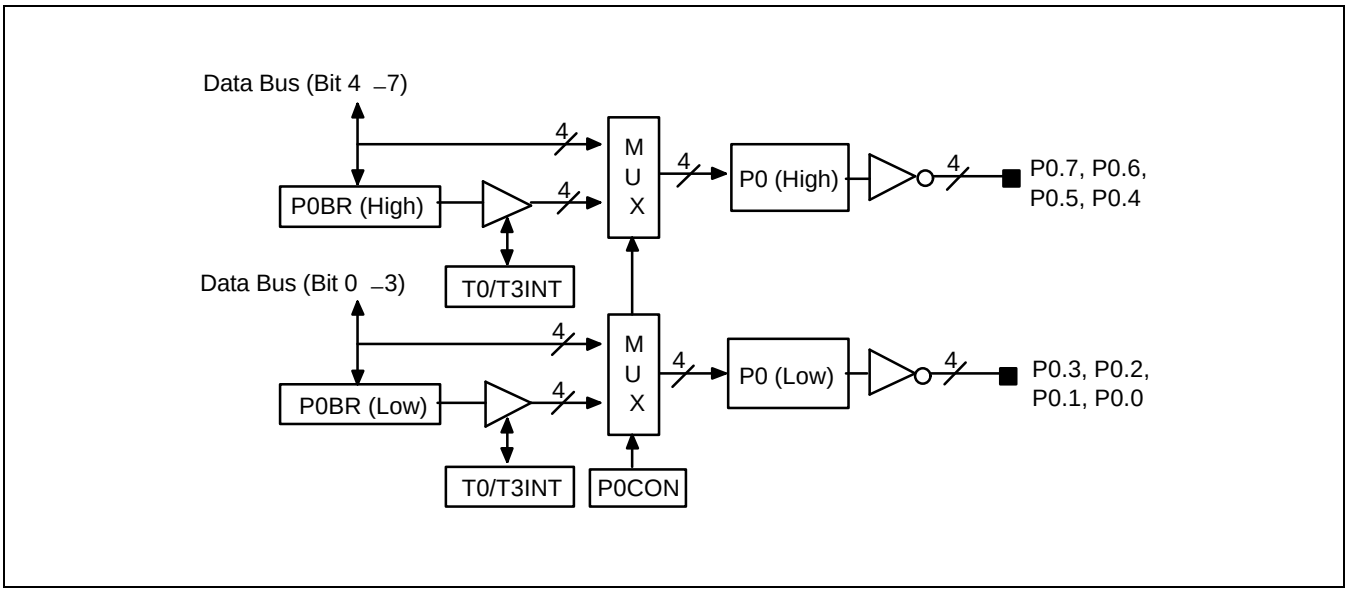


Figure 12-3. Port 0 (Real Time Output)

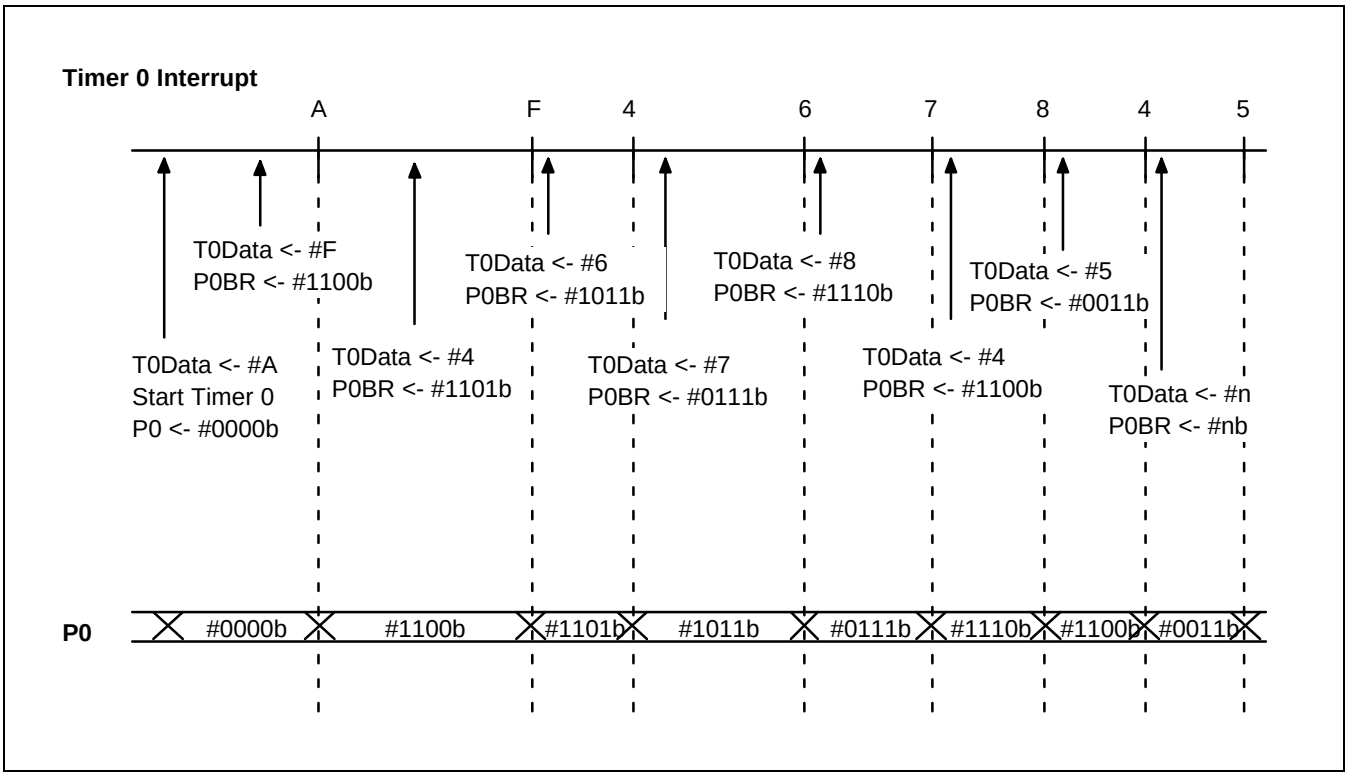


Figure 12-4. Real Time Output Example

NOTE

In MDS mode, Port 0 can not support the real time output function.

PORT 1

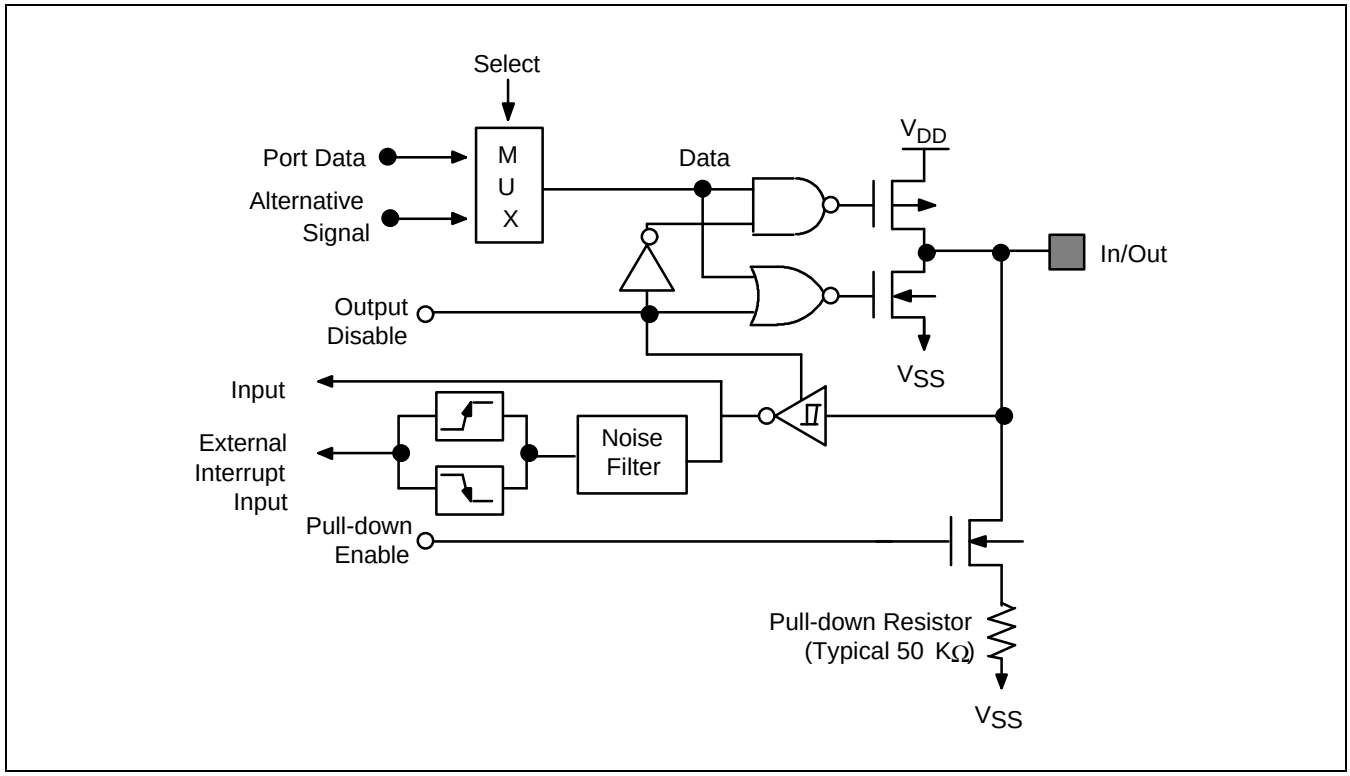


Figure 12-5. Pin Circuit Type 1 (Port 1.0/INT2/A16-1.7/INT9/A23)

Table 12-5. Port 1 Control Register Summary

Name	Bit	Setting	Description
P1CON	0, 1, 2, 3, 4, 5, 6, 7	0 or 1	Setting the corresponding bit of Port 1 0: Input or external interrupt input (INT2/3/4/5/6/7/8/9) 1: CMOS push-pull output mode
	8-15	0 or 1	Setting the Port 1 0: Normal in/output mode (P1CON.0-7 set the its corresponding bit of Port 1) 1: Address bus line mode (Do not care the values of P1CON.0-7)
P1PDR Reset Value : FFH	0, 1, 2, 3, 4, 5, 6, 7	0 or 1	Setting the corresponding pull-down resistor of Port 1. 0: Disable pull-down resistor 1: Enable pull-down resistor (When P1 is set as Address line, pull-down resistor is automatically disabled.)
EINTMOD	1, 0	0 or 1	Setting the external interrupt mode of Port 1.0 (INT2) 00: Falling edge interrupt enable 01: Rising edge interrupt enable 10: Falling or rising edge interrupt enable
	3, 2	0 or 1	Setting the external interrupt mode of Port 1.1 (INT3) 00: Falling edge interrupt enable 01: Rising edge interrupt enable 10: Falling or rising edge interrupt enable

Table 12-5. Interrupt Enable Register Summary (Continued)

Name	Bit	Setting	Description
EINTMOD	5, 4	0 or 1	Setting the external interrupt mode of Port 1.2 (INT4) 00: Falling edge interrupt enable 01: Rising edge interrupt enable 10: Falling or rising edge interrupt enable
	7, 6	0 or 1	Setting the external interrupt mode of Port 1.3 (INT5) 00: Falling edge interrupt enable 01: Rising edge interrupt enable 10: Falling or rising edge interrupt enable
	9, 8	0 or 1	Setting the external interrupt mode of Port 1.4 (INT6) 00: Falling edge interrupt enable 01: Rising edge interrupt enable 10: High level interrupt enable 11: Low level interrupt enable
	11, 10	0 or 1	Setting the external interrupt mode of Port 1.5 (INT7) 00: Falling edge interrupt enable 01: Rising edge interrupt enable 10: High level interrupt enable 11: Low level interrupt enable
	13, 12	0 or 1	Setting the external interrupt mode of Port 1.6 (INT8) 00: Falling edge interrupt enable 01: Rising edge interrupt enable 10: High level interrupt enable 11: Low level interrupt enable
	15, 14	0 or 1	Setting the external interrupt mode of Port 1.7 (INT9) 00: Falling edge interrupt enable 01: Rising edge interrupt enable 10: High level interrupt enable 11: Low level interrupt enable
EINTCON	0-7	0 or 1	Setting the external interrupt enable of Port 1.0-1.7 (INT2-9) 0: Disable External Interrupt 1: Enable External Interrupt

PORT 2

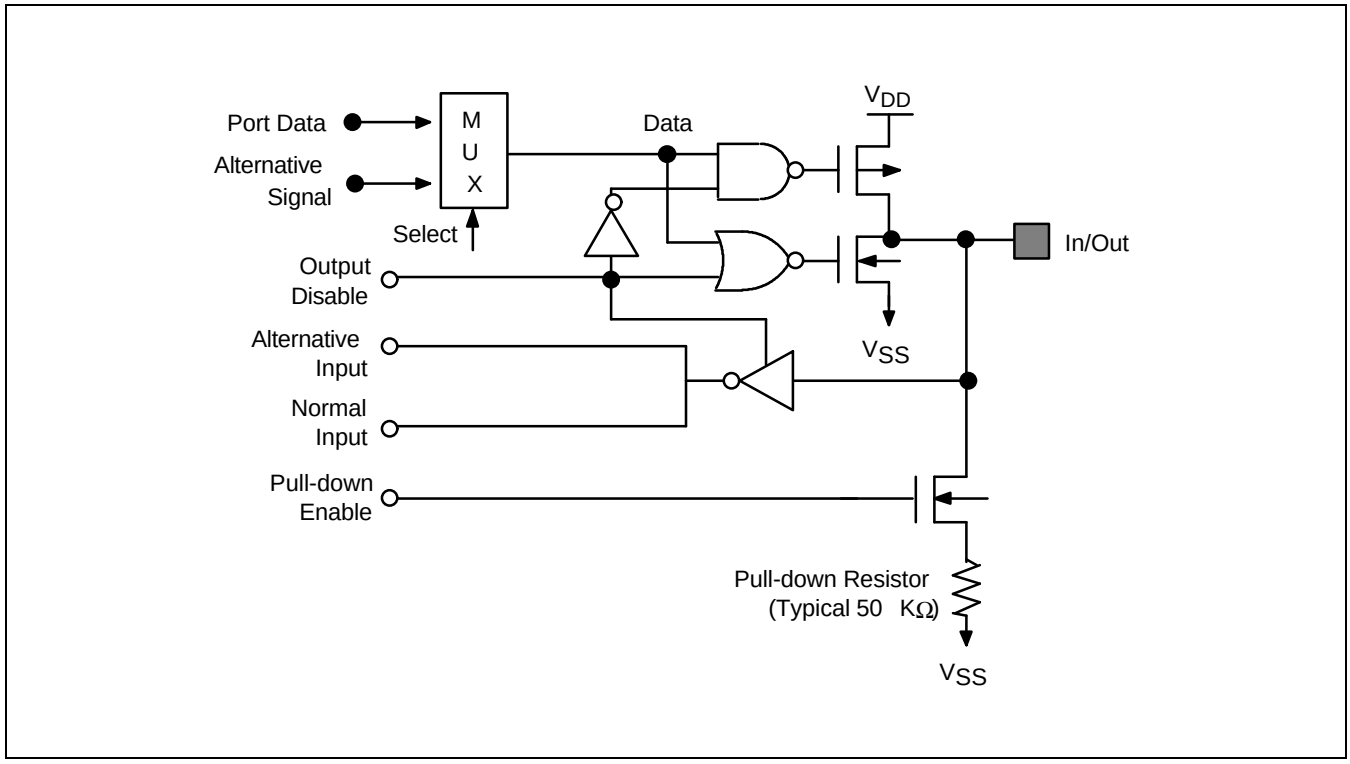


Figure 12-6. Pin Circuit Type 2 (Port 2)

Table 12-6. Port 2 Control Register Summary

Name	Bit	Setting	Description
P2CON	0/1-14/15	0 or 1	Each P2 bit Setting 00: CMOS input mode 01: CMOS push-pull output mode 10: Address bus line (A16-A23) 11: Data bus line (D8-D15)
P2PDR Reset Value : FFH	0, 1, 2, 3, 4, 5, 6, 7	0 or 1	Setting the corresponding pull-down resistor of Port 2. 0: Disable pull-down resistor 1: Enable pull-down resistor

PORT 3

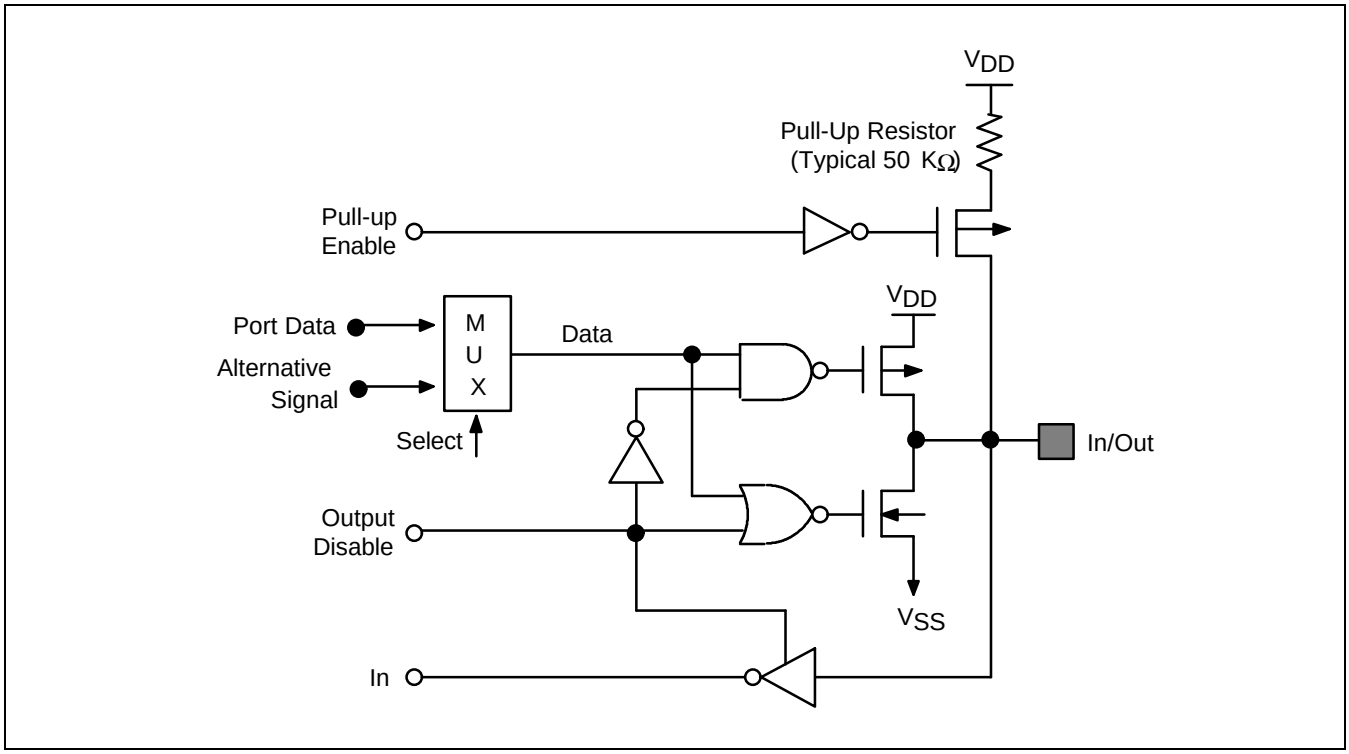


Figure 12-7. Pin Circuit Type 3 (Port 3.0, 3.2-3.7)

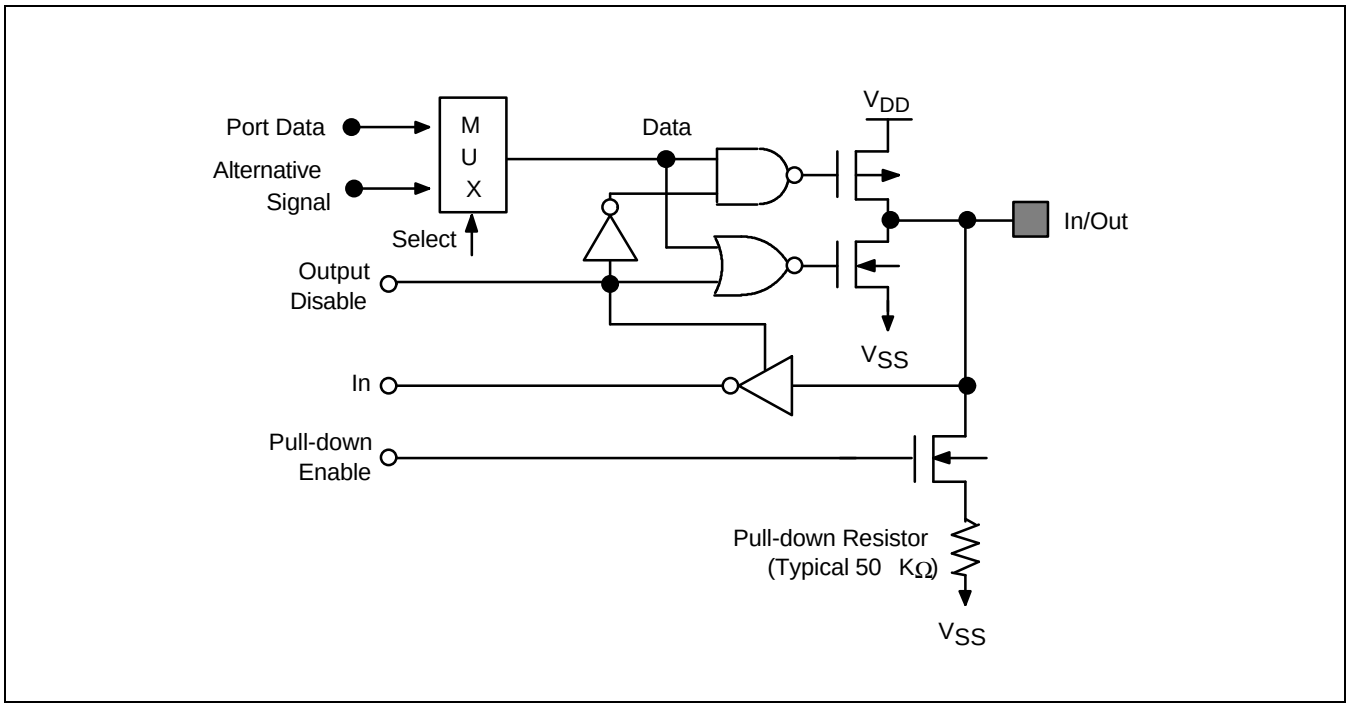


Figure 12-8. Pin Circuit Type 3-1 (Port 3.1)

Table 12-7. Port 3 Control Register Summary

Name	Bit	Setting	Description
P3CON	1, 0	0 or 1	P3.0 Setting 00: CMOS input mode 01: CMOS push-pull output mode 10: High write strobe signal (nWBE1) output for the external interface
	3, 2	0 or 1	P3.1 Setting 00: CMOS input mode 01: CMOS push-pull output mode 10: chip select signal (nCS0) output for the external interface
	5, 4	0 or 1	P3.2 Setting 00: CMOS input mode 01: CMOS push-pull output mode 10: Row address strobe signal (nRAS1) or chip select signal (nCS1) output for the external interface
	7, 6	0 or 1	P3.3 Setting 00: CMOS input mode 01: CMOS push-pull output mode 10: Row address strobe signal (nRAS2) or chip select signal (nCS2) output for the external interface
	9, 8	0 or 1	P3.4 Setting 00: CMOS input mode 01: CMOS push-pull output mode 10: Row address strobe signal (nRAS3) or chip select signal (nCS3) output for the external interface
	11, 10	0 or 1	P3.5 Setting 00: C-MOS input mode 01: C-MOS push-pull output mode 10: Column address strobe signal (nCAS0) or extra chip select signal (nECS0) output for the external interface
	13, 12	0 or 1	P3.6 Setting 00: CMOS input mode 01: CMOS push-pull output mode 10: Column address strobe signal (nCAS1) or extra chip select signal (nECS1) output for the external interface
	15, 14	0 or 1	P3.7 Setting 00: CMOS input mode 01: CMOS push-pull output mode 10: Address strobe signal (nAS) output for the external interface or Write enable signal (nSWE) output for the x16 SRAM external memory interface.
P3PUR Reset Value : FFH	0	0 or 1	Setting the corresponding pull-up resistor of Port 3.0 0: Disable pull-up resistor 1: Enable pull-up resistor
	1	0 or 1	Setting the corresponding pull-down resistor of Port 3.1 0: Disable pull-down resistor 1: Enable pull-down resistor
	2, 3, 4, 5, 6, 7	0 or 1	Setting the corresponding pull-up resistor of Port 3.2-3.7 0: Disable pull-up resistor 1: Enable pull-up resistor

NOTE: 11 is a invalid value.

PORT 4

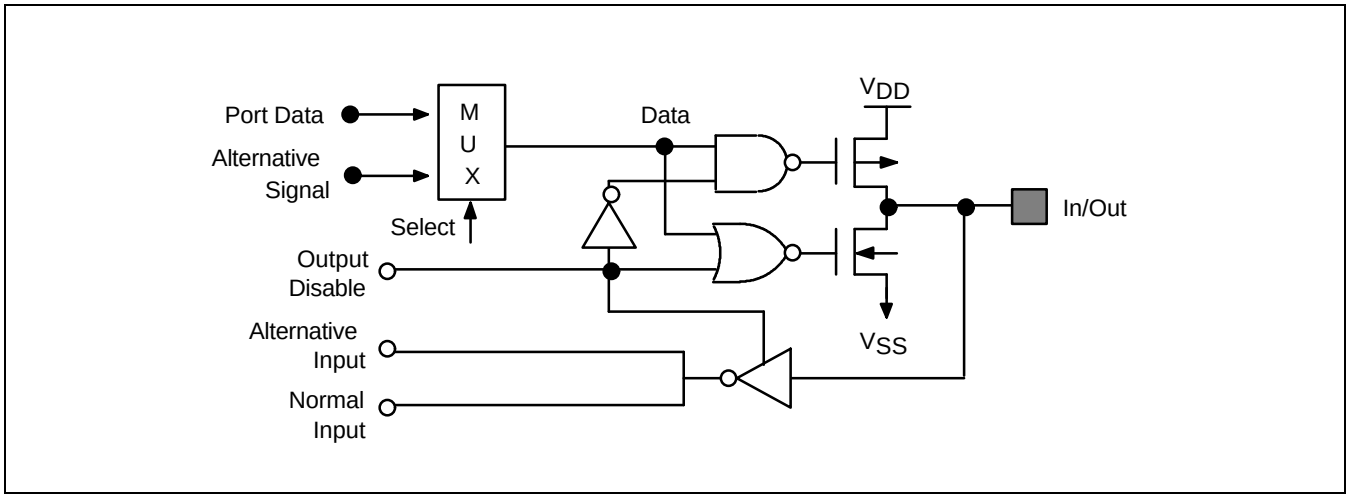


Figure 12-9. Pin Circuit Type 4 (Port 4.0-4.4)

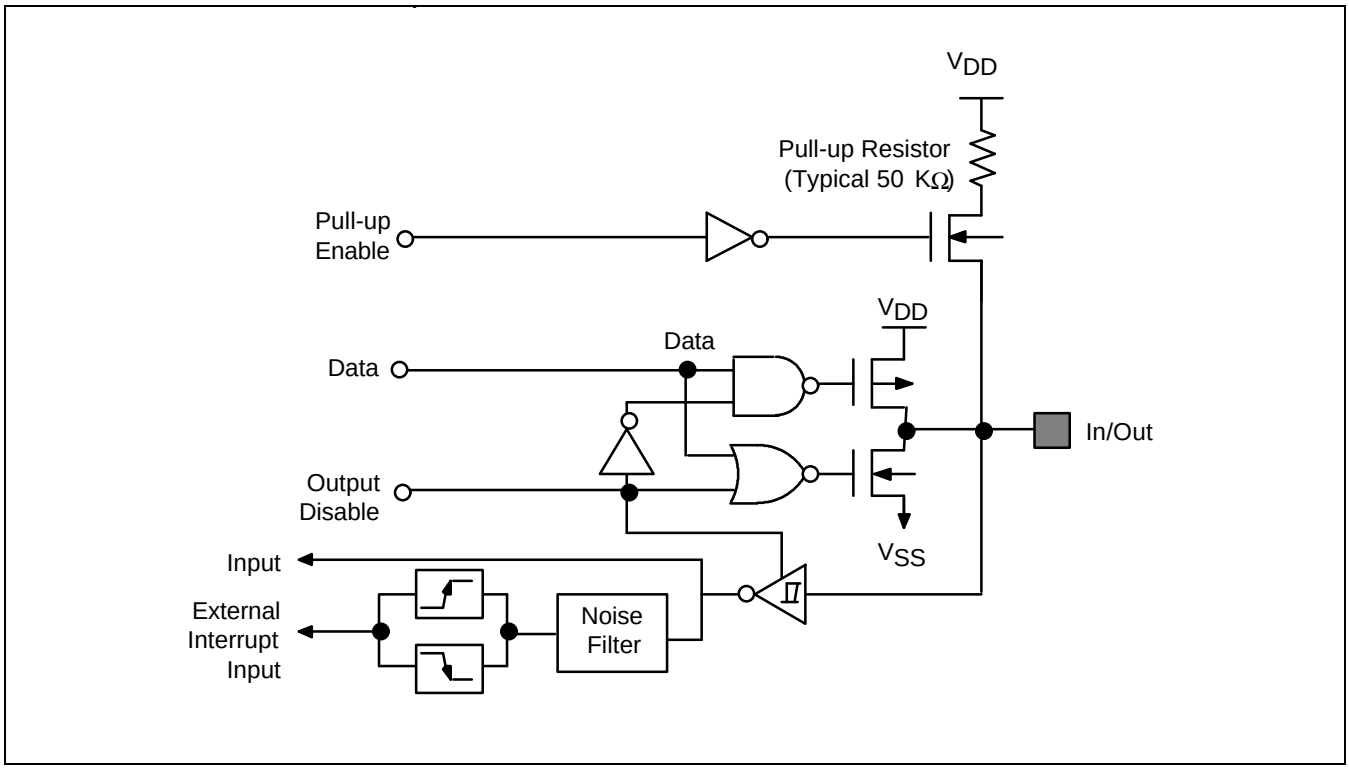


Figure 12-10. Pin Circuit Type 4-1 (P4.5/INT1, P6.7/INT0)

Table 12-8. Port 4 Control Register Summary

Name	Bit	Setting	Description
P4CON	1, 0	0 or 1	P4.0 Setting 00: CMOS input mode 01: CMOS push-pull output mode 10: Wait signal (nWAIT) input for the external interface 11: Busy signal (nBusy) input or output for the serial I/O 0,1 synchronization.
	3, 2	0 or 1	P4.1 Setting 00: CMOS input mode 01: CMOS push-pull output mode 10 : External DMA request input (nXDREQ0)
	5, 4	0 or 1	P4.2 Setting 00: CMOS input mode 01: CMOS push-pull output mode 10: External DMA acknowledge output (nXDACK0)
	7, 6	0 or 1	P4.3 Setting 00: CMOS input mode 01: CMOS push-pull output mode 10: External DMA request input (nXDREQ1)
	9, 8	0 or 1	P4.4 Setting 00: CMOS input mode 01: CMOS push-pull output mode 10: External DMA acknowledge output (nXDACK1)
	10	0 or 1	P4.5 Setting 0: Schmitt input mode or external interrupt input (INT1) 1: C-MOS push-pull output mode
EINTMOD	17, 16	0 or 1	Setting the external interrupt mode of Port 6.7 (INT0) 00: Falling edge interrupt enable 01: Rising edge interrupt enable 10: High level interrupt enable 11: Low level interrupt enable
	19, 18	0 or 1	Setting the external interrupt mode of Port 4.5 (INT1) 00: Falling edge interrupt enable 01: Rising edge interrupt enable 10: High level interrupt enable 11: Low level interrupt enable
EINTCON	9	0 or 1	Setting the external interrupt enable of Port 6.7 (INT0) 0: Disable External Interrupt 1: Enable External Interrupt
	8	0 or 1	Setting the external interrupt enable of Port 4.5 (INT1) 0: Disable External Interrupt 1: Enable External Interrupt
P4PUR	0	0 or 1	Setting the pull-up resistor of Port 4.5 0: Disable pull-up resistor 1: Enable pull-up resistor

NOTE: If P4.0 is configured as nBusy signal for serial I/O interface synchronization, it will be changed to N-channel open drain output and input mode. In DMA mode, nBusy pin function is very useful to check the readiness for SIO operations of the KS17C4000 or the external device.

PORT 5

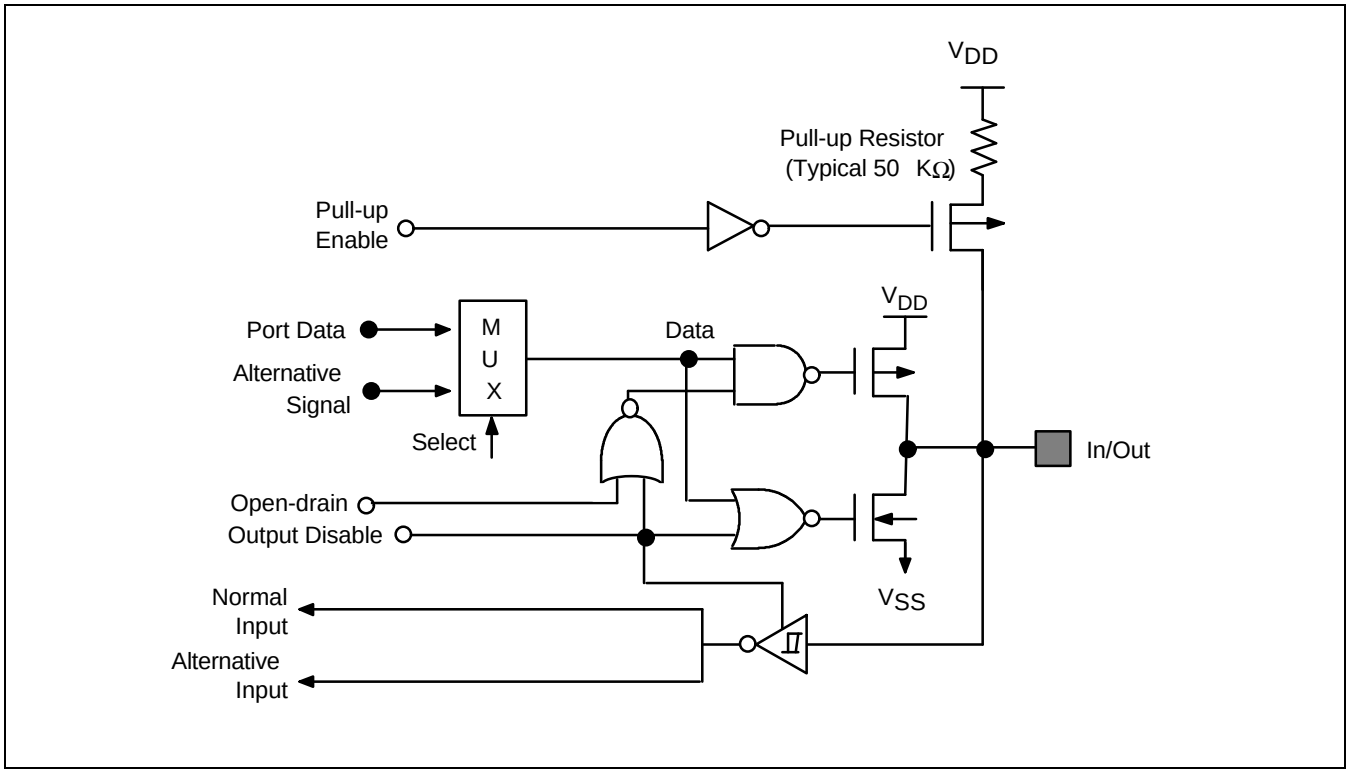


Figure 12-11. Pin Circuit Type 5 (Port 5)

Table 12-9. Port 5 Control Register Summary

Name	Bit	Setting	Description
P5CON	1, 0	0 or 1	P5.0 Setting 00: Schmitt input mode or serial input (Rx) for UART 01: CMOS push-pull output mode 10: N-Channel open drain output mode
	4, 3, 2	0 or 1	P5.1 Setting 000: Schmitt input mode 001: CMOS push-pull output mode 011: N-Channel open drain output mode 101: CMOS push-pull serial output (Tx) for UART 111: N-Channel open drain serial output (Tx) for UART
	7, 6, 5	0 or 1	P5.2 Setting 000: Schmitt input mode. 001: CMOS push-pull output mode 011: N-Channel open drain output 101: CMOS push-pull serial data output (SO0) for SIO0 111: N-Channel open drain serial data output (SO0) for SIO0

Table 12-9. Port 5 Control Register Summary (Continued)

Name	Bit	Setting	Description
P5CON	9, 8	0 or 1	P5.3 Setting 00: Schmitt input mode or serial data input (SI0) for SIO0 01: CMOS push-pull output. 10: N-Channel open drain output mode
	12, 11, 10	0 or 1	P5.4 Setting 000: Schmitt input mode or serial clock input(SCK0) for SIO0 or UART 001: CMOS push-pull output mode 010: N-Channel open drain output mode 101: CMOS push-pull serial clock output (SCK0) for SIO0 110: N-Channel open drain serial clock output (SCK0) for SIO0
	15, 14, 13	0 or 1	P5.5 Setting 000: Schmitt input mode or serial clock input (SCK1) for SIO 001: CMOS push-pull output mode 010: N-Channel open drain output mode 101: CMOS push-pull serial clock output (SCK1) for SIO1 110: N-Channel open drain serial clock output (SCK1) for SIO1
	18, 17, 16	0 or 1	P5.6 Setting 000: Schmitt input mode 001: C-MOS push-pull output mode 010: N-Channel open drain output mode 101: CMOS push-pull serial data output (SO1) for SIO1 110: N-Channel open drain serial data output (SO1) for SIO1
	20, 19	0 or 1	P5.7 Setting 00: Schmitt input mode or serial data input (SI1) for SIO1 01: CMOS push-pull output mode 10: N-Channel open drain output mode
P5PUR	0, 1, 2, 3, 4, 5, 6, 7	0 or 1	Setting the corresponding pull-up resistor of Port 5. 0: Disable pull-up resistor 1: Enable pull-up resistor

NOTE: 11 is a Invalid value.

PORT 6

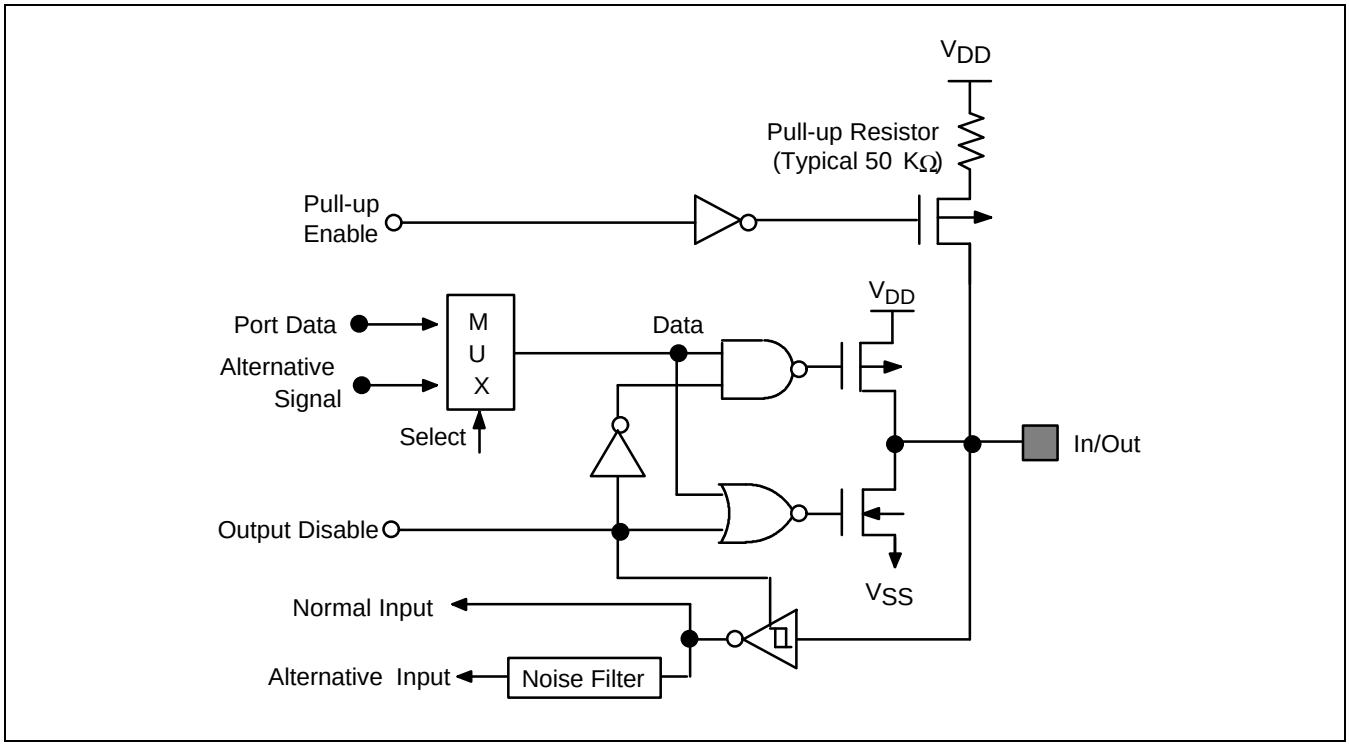


Figure 12-12. Pin Circuit Type 6 (Port 6)

Table 12-10. Port 6 Control Register Summary

Name	Bit	Setting	Description
P6CON	0	0 or 1	P6.0 Setting 0: Schmitt input mode or capture (T0_CAP) or clock (T0_CK) input for Timer 0 1: CMOS push-pull output mode
	1	0 or 1	P6.1 Setting 0: Schmitt input mode or capture (T1_CAP) or clock (T1_CK) input for Timer 1 1: CMOS push-pull output mode
	2	0 or 1	P6.2 Setting 0: Schmitt input mode or capture (T2_CAP) or clock (T2_CK) input for Timer 2 1: CMOS push-pull output mode
	3	0 or 1	P6.3 Setting 0: Schmitt input mode or clock (T3_CK) input for Timer 3 1: CMOS push-pull output mode
	4	0 or 1	P6.4 Setting 0: Schmitt input mode or clock (T4_CK) input for Timer 4 1: CMOS push-pull output mode

Table 12-10. Port 6 Control Register Summary (Continued)

Name	Bit	Setting	Description
P6CON	6, 5	0 or 1	P6.5 Setting 00: Schmitt input mode or capture (T3_CAP) input for Timer 3 01: CMOS push-pull output mode 10: CMOS push-pull PWM/Out (T3PWM/T3Out) output for Timer 3.
	8, 7	0 or 1	P6.6 Setting 00: Schmitt input mode or capture (T4_CAP) input for Timer 4 01: CMOS push-pull output mode 10: CMOS push-pull PWM/Out (T4PWM/T4Out) output for Timer 4.
	9	0 or 1	P6.7 Setting 0: Schmitt input mode or external interrupt input (INT0) 1: CMOS push-pull output mode
P6PUR	0, 1, 2, 3, 4, 5, 6, 7	0 or 1	Setting the corresponding pull-up resistor of Port 6. 0: Disable pull-up resistor 1: Enable pull-up resistor

PORT 7

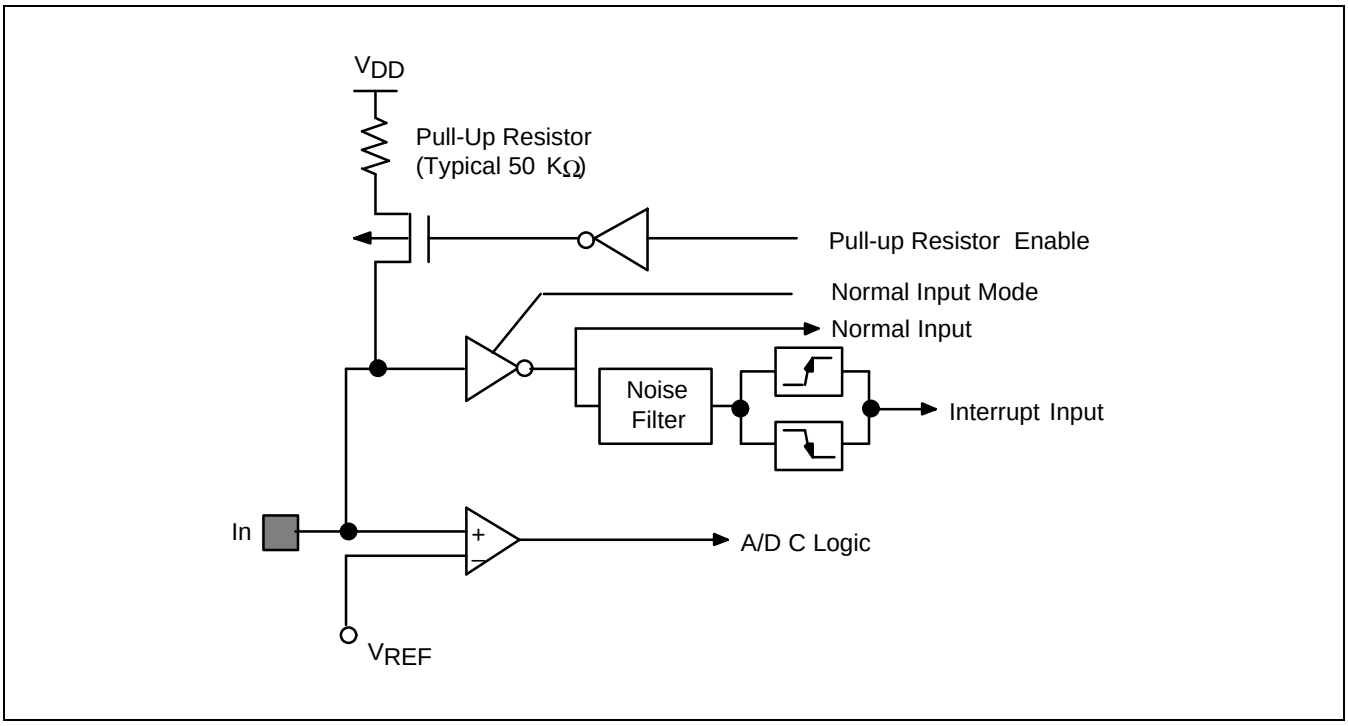


Figure 12-13. Pin Circuit Type 7 (P7.0/ADC0/INT10-P7.2/ADC2/INT12)

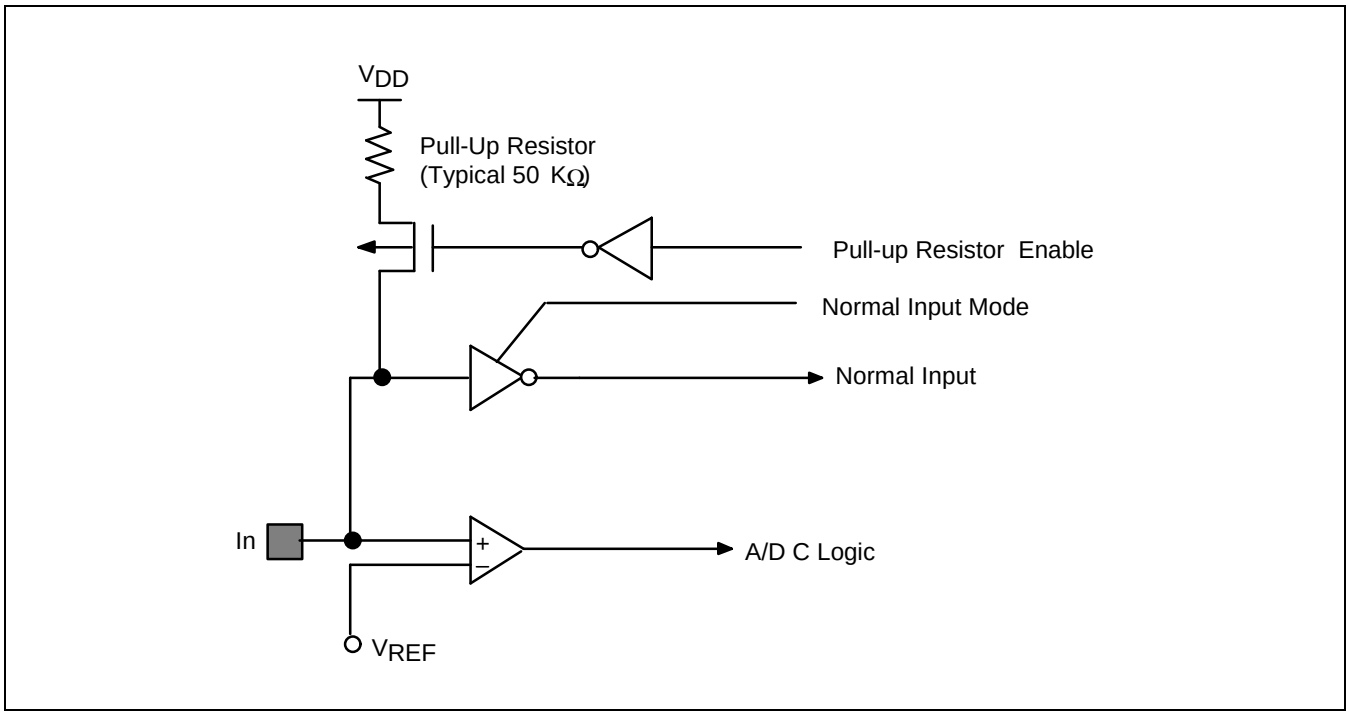


Figure 12-14. Pin Circuit Type 7-1 (P7.3/ADC3-P7.7/ADC7)

Table 12-11. Port7 Control Register Summary

Name	Bit	Setting	Description
P7CON	0, 1, 2, 3, 4, 5, 6, 7	0 or 1	P7.0, P7.1, P7.2, P7.3, P7.4, P7.5, P7.6, P7.7 Setting 0: CMOS input mode 1: A/D C input mode.
P7PUR	0, 1, 2, 3, 4, 5, 6, 7	0 or 1	Setting the corresponding pull-up resistor of Port 7 0: Disable pull-up resistor 1: Enable pull-up resistor
EINTMOD	21, 20, 23, 22, 25, 24	0 or 1	Setting the external interrupt mode of P7.0, P7.1, or P7.2 (INT10-12) 00: Falling edge interrupt enable 01: Rising edge interrupt enable 10: Falling or rising edge interrupt enable 11: Invalid value
EINTCON	10-12	0 or 1	Setting the external interrupt enable of P7.0, P7.1, or P7.2 (INT10-12) 0: Disable External Interrupt 1: Enable External Interrupt

ADDRESS AND DATA LINE (A0-A15, D0-D7)

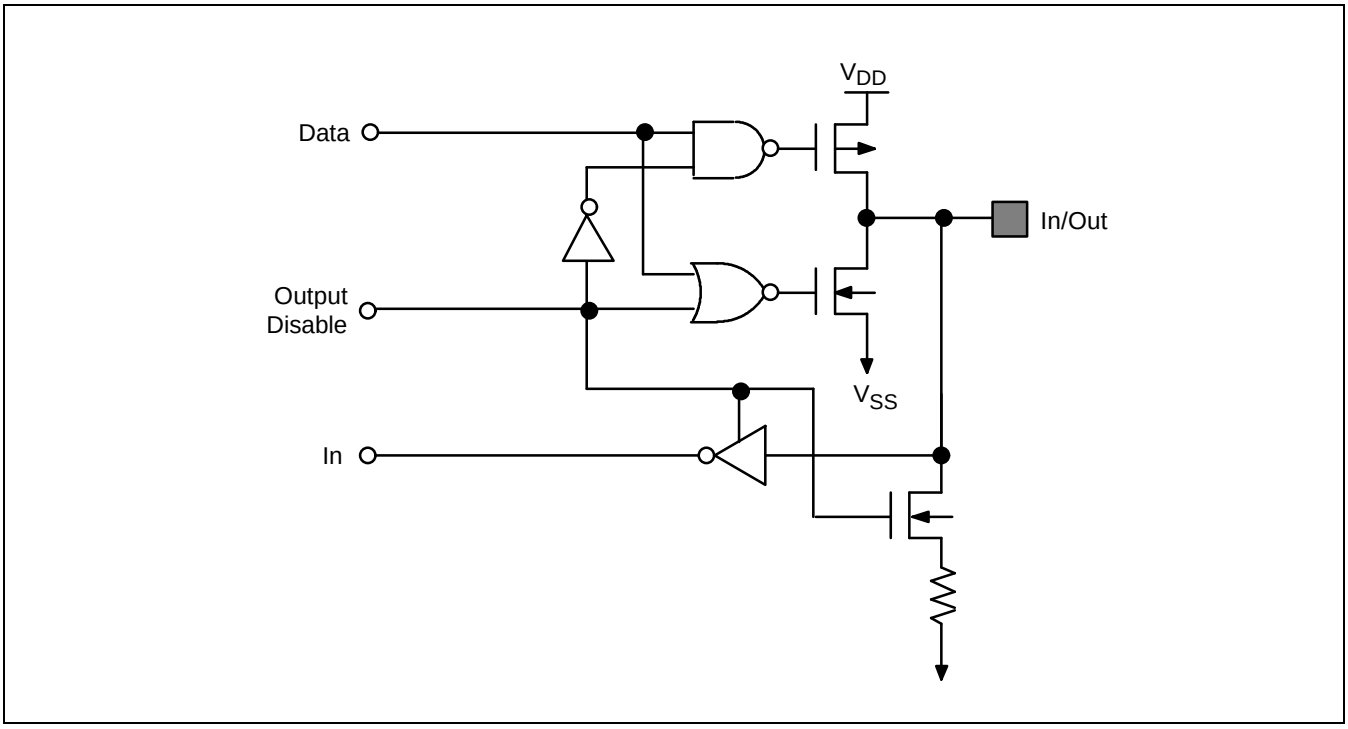


Figure 12-15. Pin Circuit Type 8 (D0-D7)

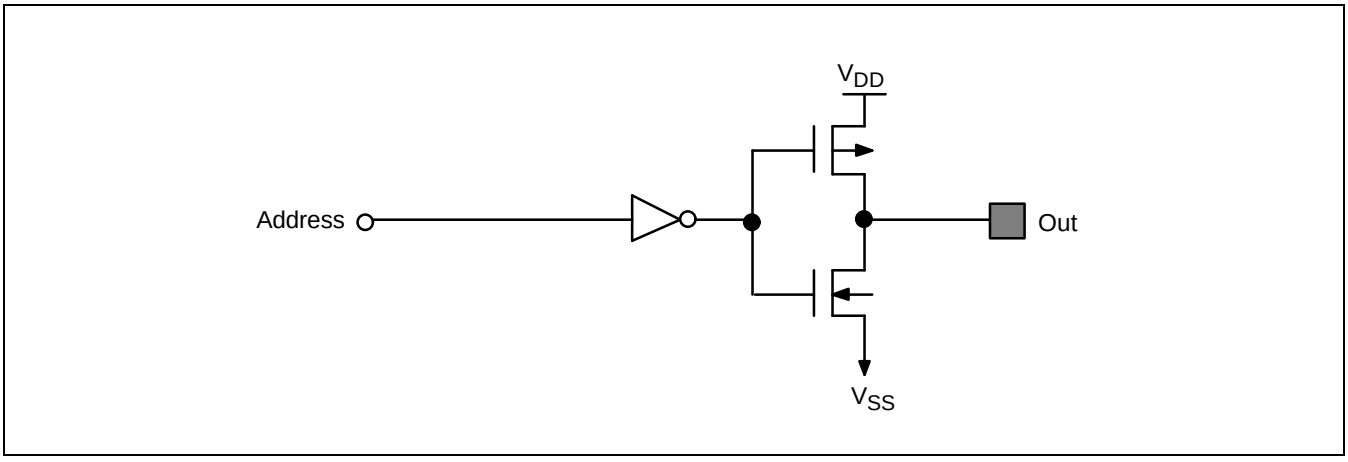


Figure 12-16. Pin Circuit Type 8-1 (A0-A15, nRD, nWBE0, nWREXP)

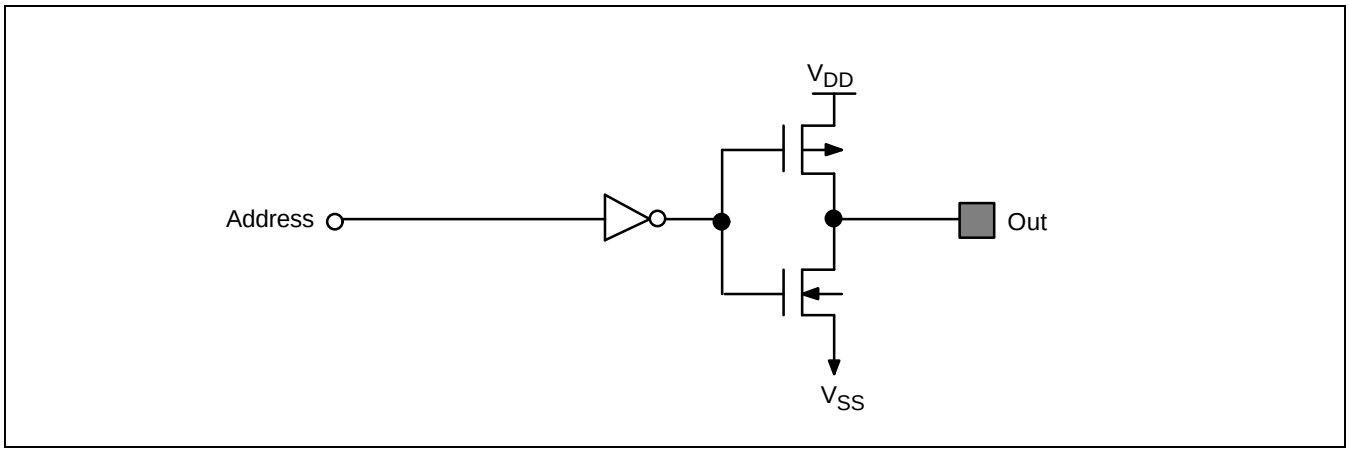


Figure 12-17. Pin Circuit Type 8-2 (A8/A16-A15/A23)

RESET

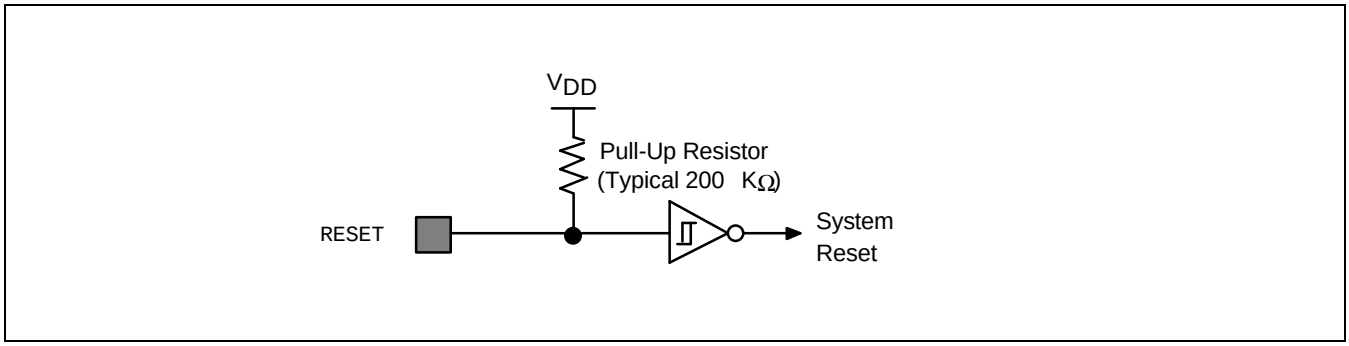


Figure 12-18. Pin Circuit Type 9 (RESET)

MEMORY INTERFACE SIGNAL CONNECTION METHOD

Table 12-12. Control Pin Descriptions

Pin Name	Memory Type	Description
nRD	Any type	Memory read strobe signal
nWBE0	Single x8 Flash ROM/EEPROM/DRAM/SRAM	Memory write strobe signal
	Two x8 Flash ROM/EEPROM/DRAM/SRAM	Memory lower byte write strobe signal
	Single x16 SRAM	Memory lower byte signal
nWBE1	Two x8 Flash ROM/EEPROM/DRAM/SRAM	Memory upper byte write strobe signal
	Single x16 SRAM	Memory upper byte signal
nSWE	x16 SRAM	Memory write strobe signal

MEMORY CONNECTION EXAMPLE

Table 12-13. Pin Connection Examples for Memory Types

Memory Configuration Type	MCU Pin	Memory pin
x8/x16 ROM	nRD	nOE
(MCU data bus: x8/x16)	Vdd	(nWE)
One x8 Flash ROM/EEPROM	nRD	nOE
or one x8 SRAM (MCU x8)	nWBE0	nWE
Two x8 Flash ROM/EEPROM	nRD	nOE
or two x8 SRAM	nWBE0	nWE of lower byte
(MCU data bus: x16)	nWBE1	nWE of upper byte
x16 SRAM	nRD	nOE
(MCU data bus: x16)	nWBE0	nLB
	nWBE1	nUB
	nSWE	nWE
x8 DRAM	nRD	nOE
(MCU data bus: x8)	nRAS	nRAS
	nCAS0	nCAS
	nWBE0	nWE
Two x8 DRAM	nRD	nOE
(MCU data bus: x16)	nRAS	nRAS
	nCAS0	nCAS of lower byte
	nCAS1	nCAS of upper byte
	nWBE0	nWE

Table 12-13. Pin Connection Examples for Memory Types (Continued)

Memory Configuration Type	MCU Pin	Memory pin
x16 DRAM (MCU data bus: x16)	nRD	nOE
	nRAS	nRAS
	nCAS0	nLCAS
	nCAS1	nUCAS
	nWBE0	nWE
Two x16 DRAM (MCU data bus: x16)	nRD	nOE
	nRAS1	nRAS of lower bank
	nRAS2	nRAS of upper bank
	nCAS0	nLCAS (Lower byte)
	nCAS1	nUCAS (Upper bytes)
	nWBE0	nWE

MEMORY CONFIGURATION EXAMPLE

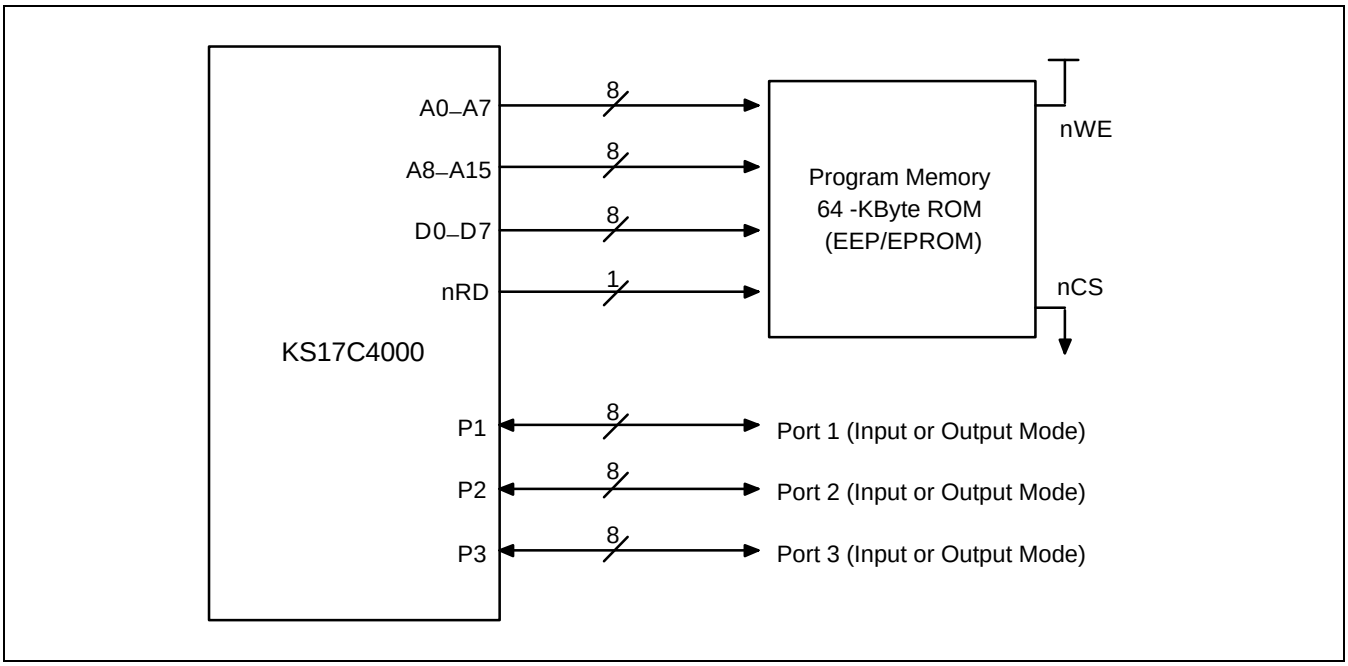


Figure 12-19. 64K × 8bit PGM Memory Only

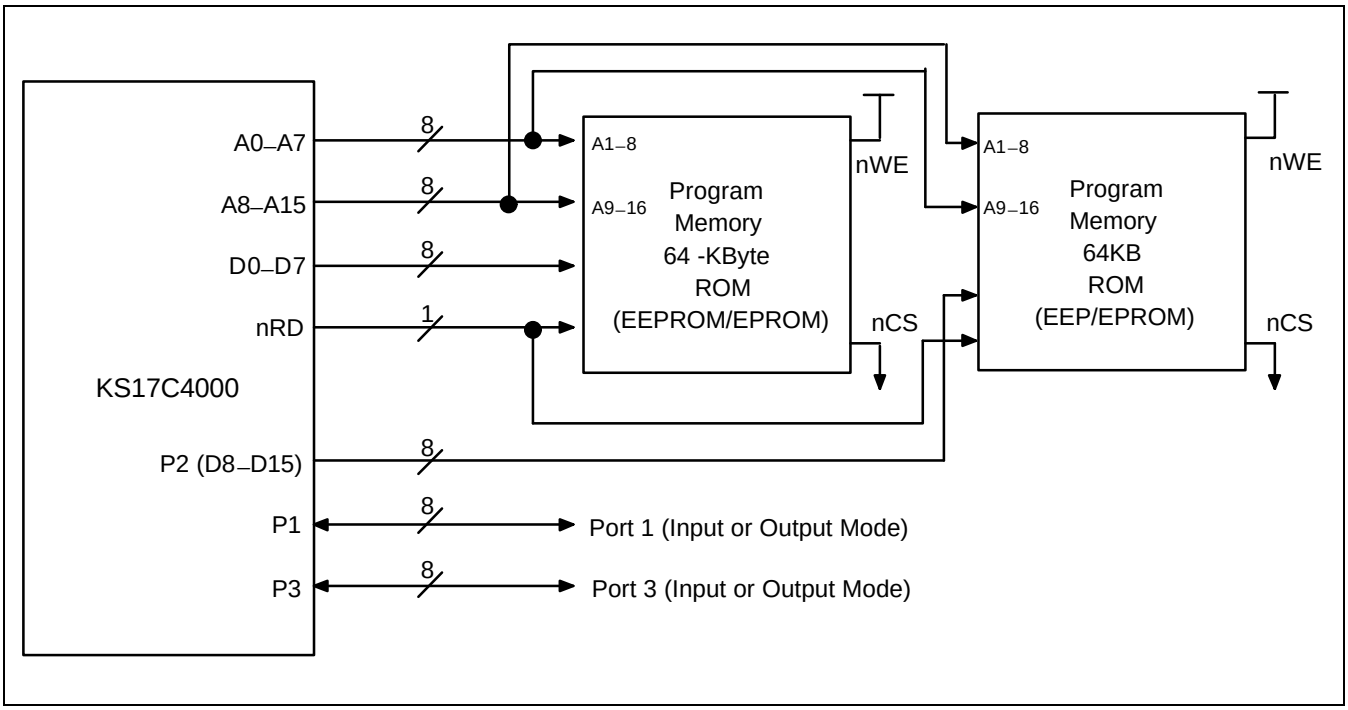


Figure 12-20. 64K × 16bit PGM Memory Only

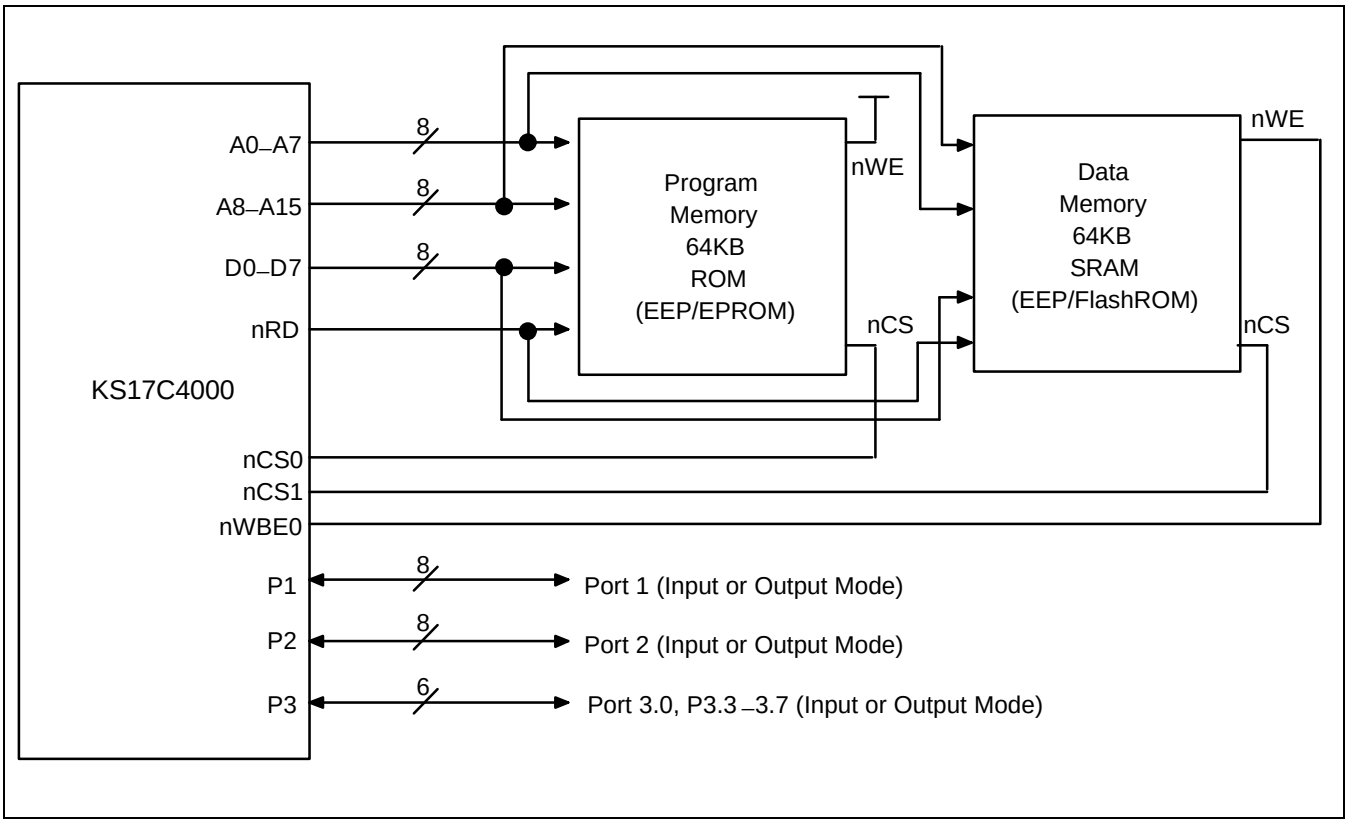


Figure 12-21. 64K × 8bit PGM & 64K × 8bit Data Memory

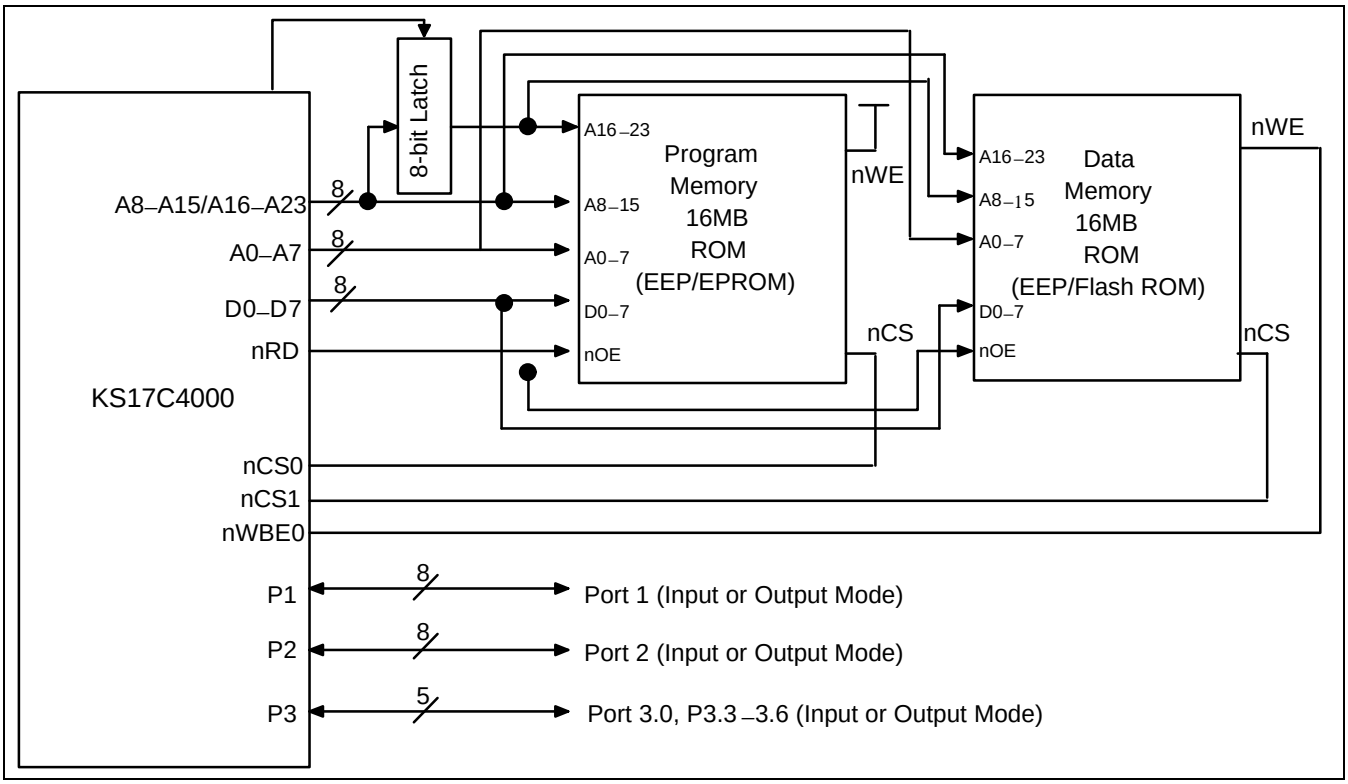


Figure 12-22. 16M × 8bit PGM & 16M × 8bit Data Memory

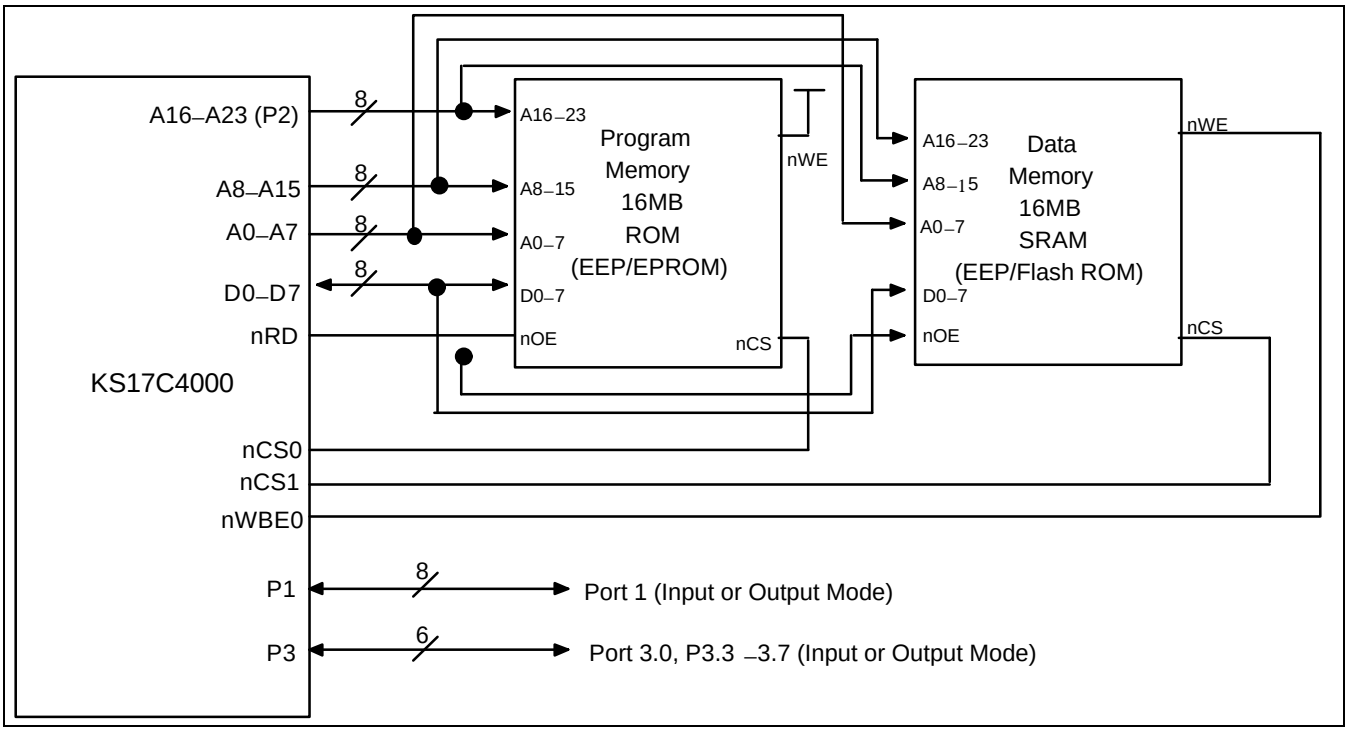


Figure 12-23. 16M × 8bit PGM & 16M × 8bit Data Memory

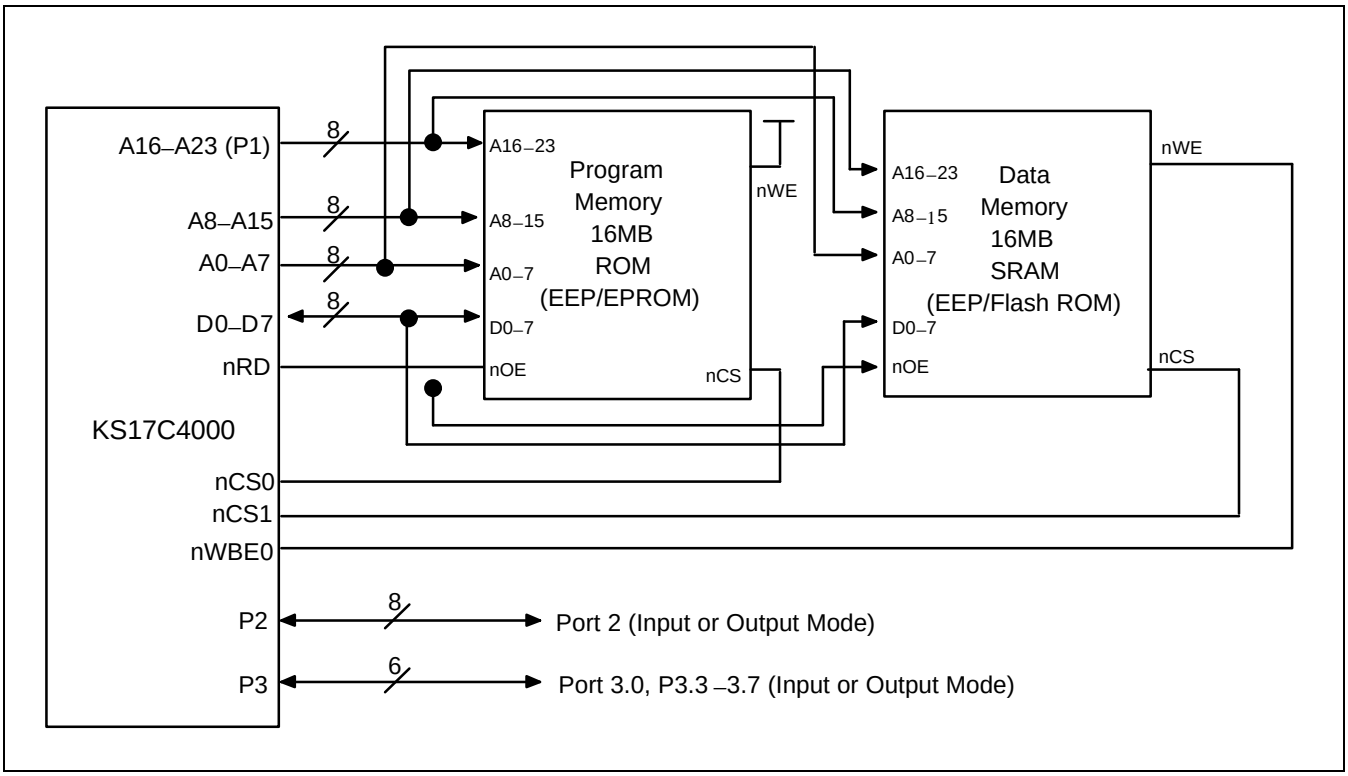


Figure 12-24. 16M × 8bit PGM & 16M × 8bit Data Memory

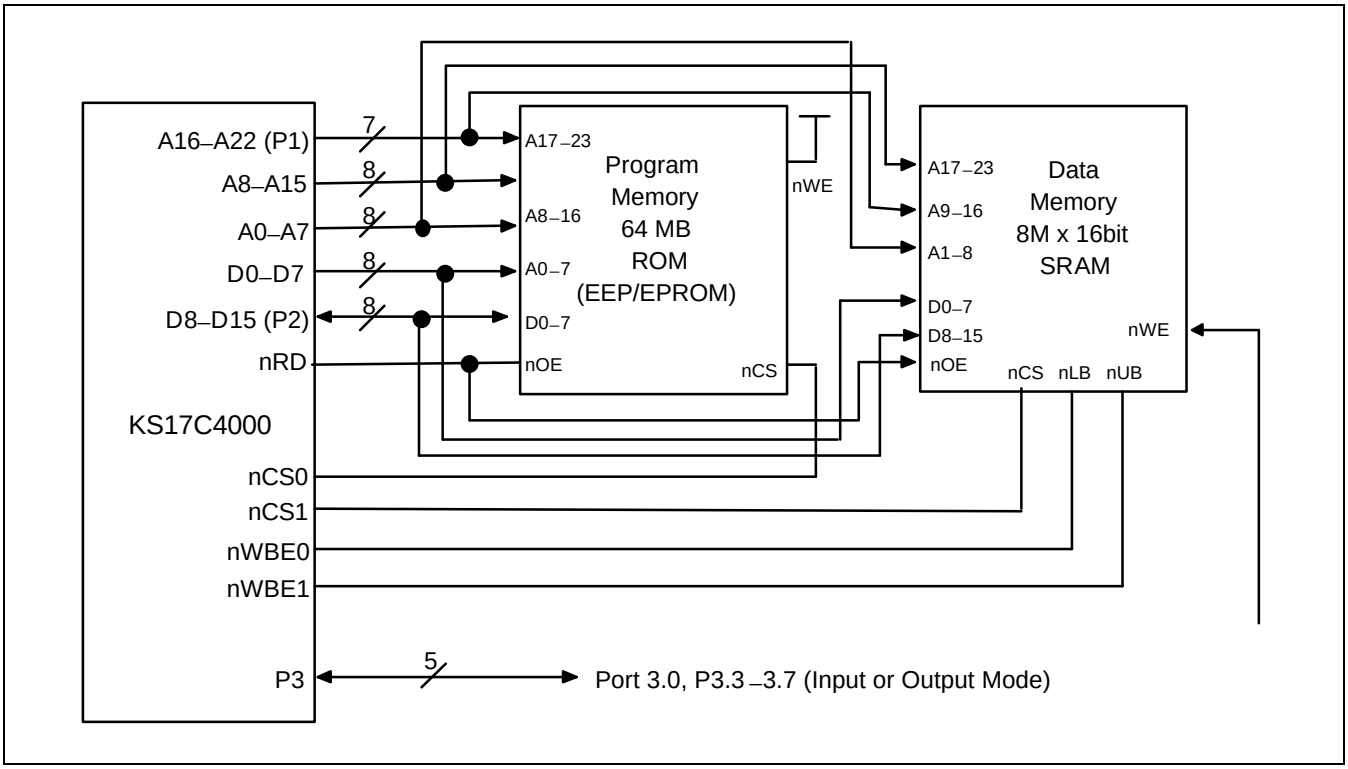


Figure 12-25. 8M × 16bit PGM and two 8M × 8bit Data Memory

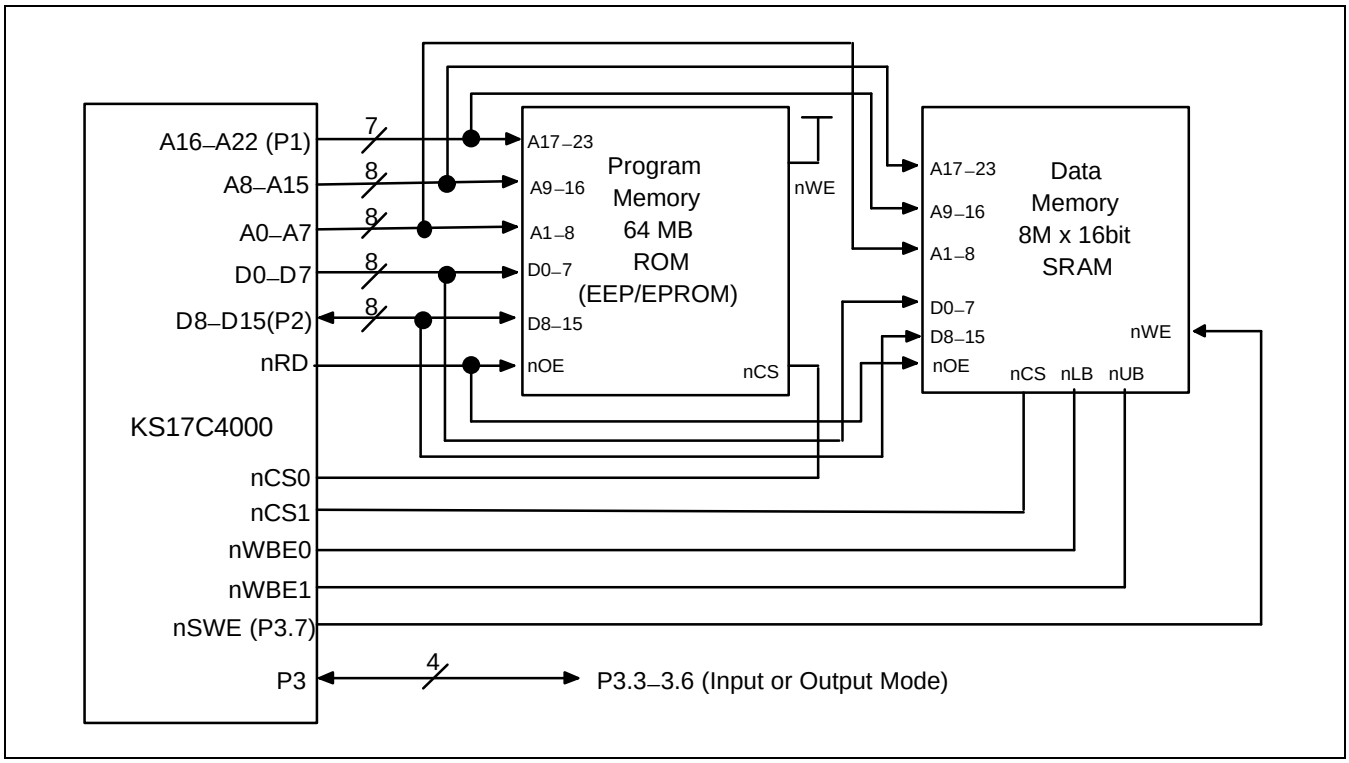


Figure 12-26. 8M × 16bit PGM and 8M × 16bit Data Memory (SRAM)

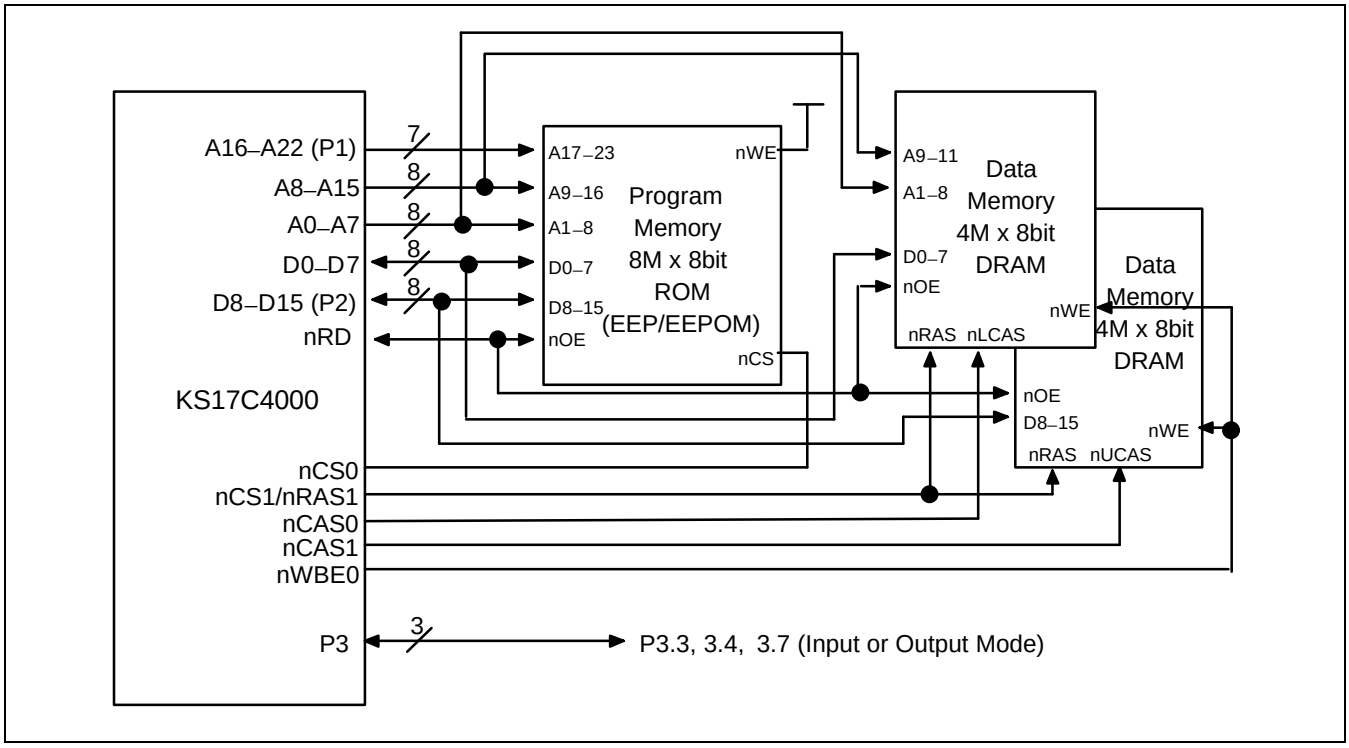


Figure 12-27. 8M × 8bit PGM and two 4M × 8bit Data Memory (DRAM)

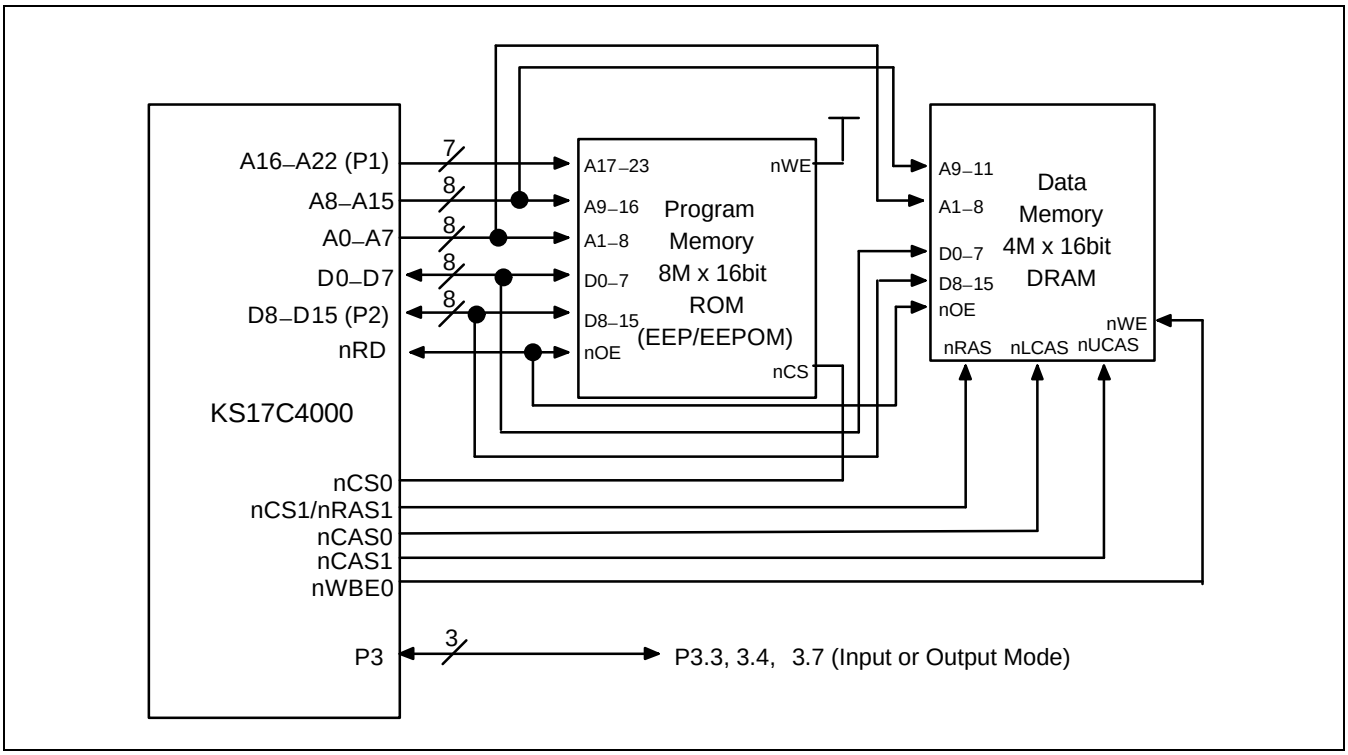


Figure 12-28. 8M × 16bit PGM and 4M × 16bit Data Memory (DRAM)

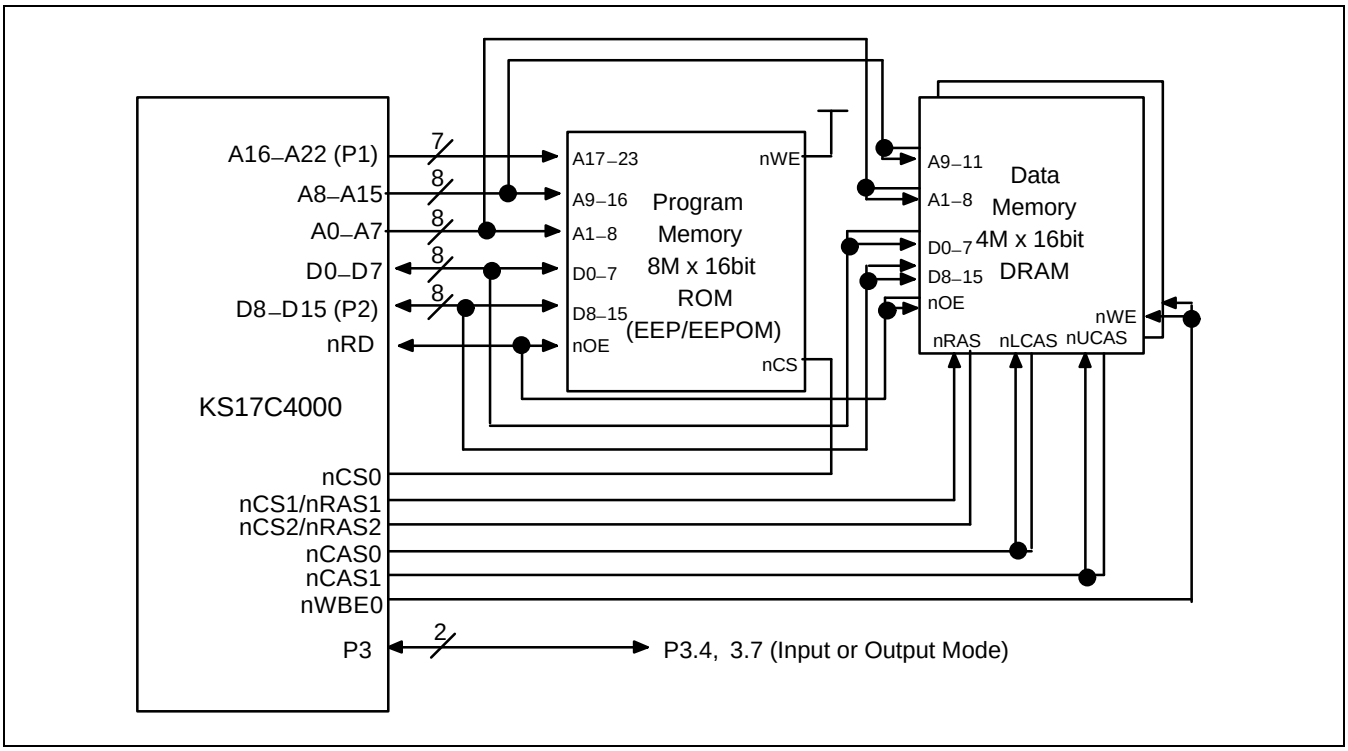


Figure 12-29. 8M × 16bit PGM and two 4M × 16bit Data Memory (DRAM)

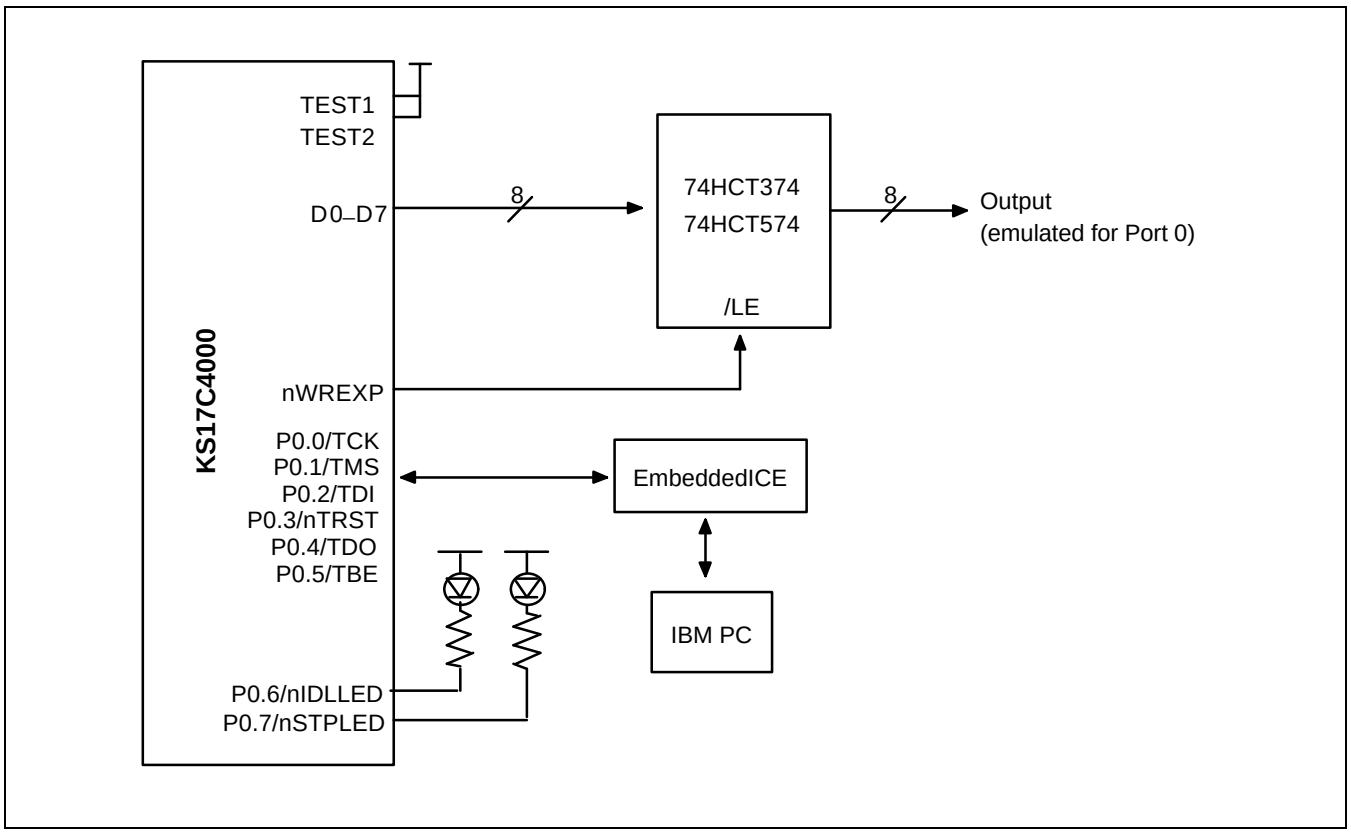


Figure 12-30. Emulation for P0 by nWREXP signal in MDS mode.

NOTE

In MDS mode, Port 0 output mode is replaced with the emulation port issued by nWREXP. However, real time output mode or input mode of port 0 are not supported by emulation port. Those port modes can be just supported by normal operating mode, not MDS mode.

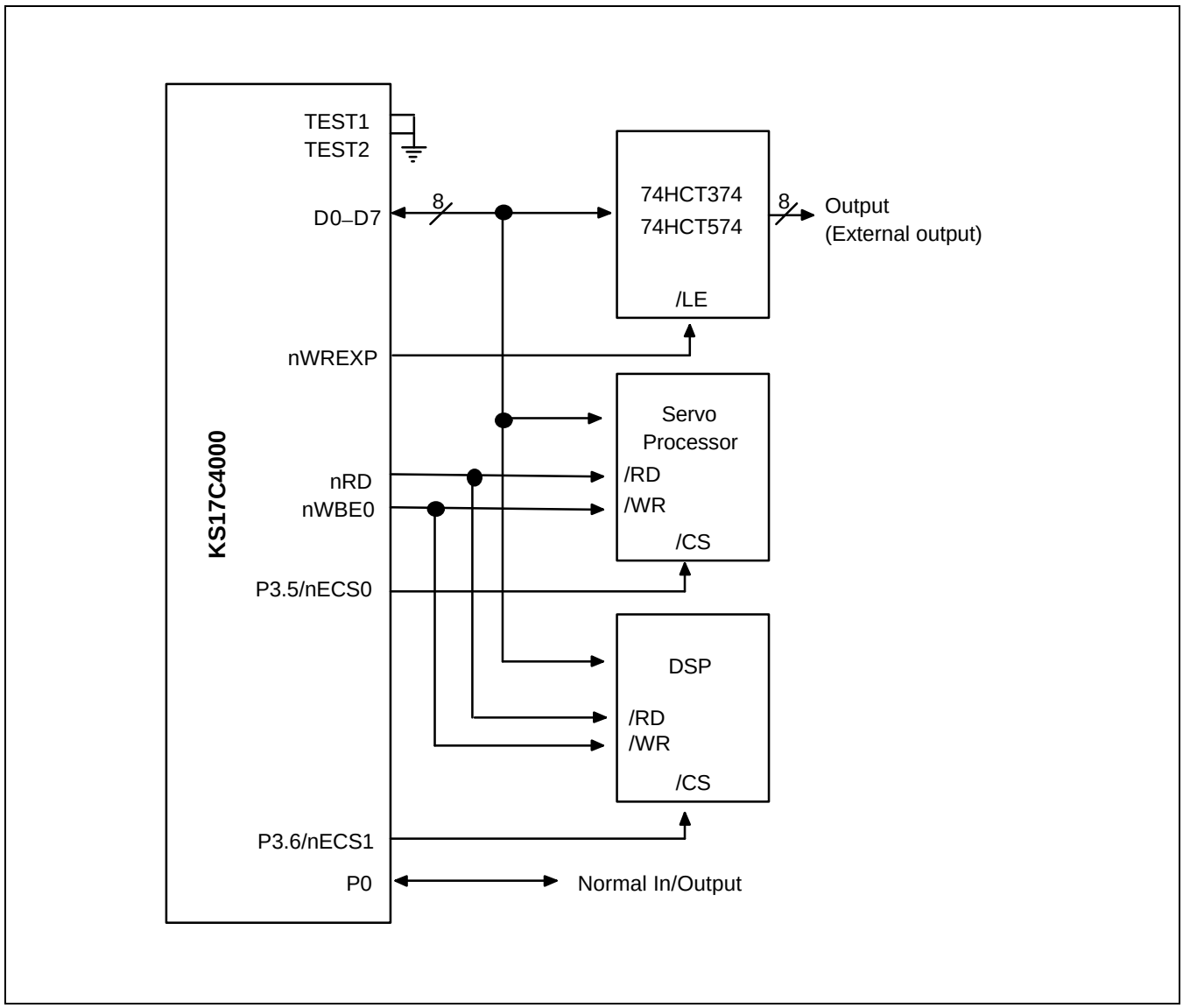


Figure 12-31. nWREXP, nECS0, and nECS1 Application Example in Normal Mode

SYSTEM RESET

OVERVIEW

During a power-on reset, the voltage at V_{DD} goes to High level and the RESET pin is forced to Low level. The RESET signal is input through a Schmitt trigger circuit where it is then synchronized with the CPU clock. This procedure brings KS17C4000 into a known operating status.

To allow time for internal CPU clock oscillation to stabilize, the RESET pin must be held to Low level for a minimum time interval after the power supply comes within tolerance. The minimum required oscillation stabilization time for a reset operation is 1 millisecond.

Whenever a reset occurs during normal operation (that is, when both V_{DD} and RESET are High level), the RESET pin is forced Low and the reset operation starts. All system and peripheral control registers are then reset to their default hardware values (see Table 12-14).

In summary, the following sequence of events occurs during a reset operation:

— All interrupts are disabled.

— The watchdog function (basic timer) is enabled.

— Ports 0-7 are set to input mode.

— Peripheral control and data registers are disabled and reset to their default hardware values.

— The program counter (PC) is loaded with the program reset address in the ROM, 000 0000H.

— When the programmed oscillation stabilization time interval has elapsed, the instruction stored in ROM location 000 0000H is fetched and executed.

NORMAL 8-BIT DATA BUS BOOT ROM MODE

The normal 8-bit bus boot ROM mode is invoked when TEST2 and TEST1 pins are tied to V_{SS}. A reset enables access to the 64-Kbyte boot ROM area. All port is set to input mode and all port 1, 2, and 3 resistor are enabled.

NORMAL 16-BIT DATA BUS BOOT ROM MODE

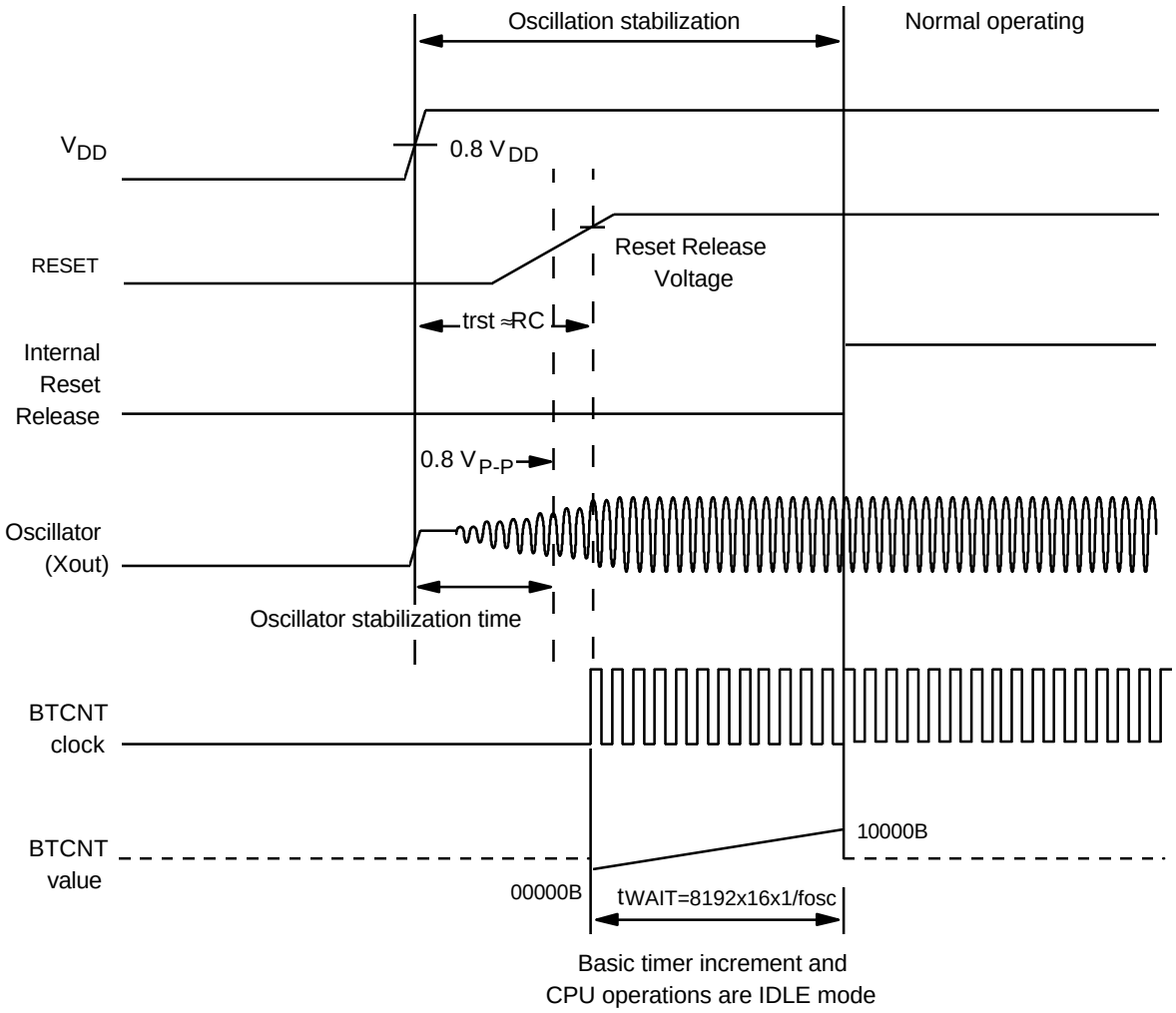
To configure x16 data bus boot ROM mode, you must apply a constant V_{DD} level to the TEST1 pin. Assuming the TEST1 pin is held to high level (5 V) when a reset occurs, 16-bit bus boot ROM mode is entered and the external interface is configured automatically. Port 2 is configured to D8-D15 bus.

NOTE

To program the duration of the oscillation stabilization interval, you make the appropriate settings to the basic timer control register, BTCON, *before* entering Stop mode. Also, if you do not want to use the basic timer watchdog function (which causes a system reset if a basic timer counter overflow occurs), you can disable it by writing '10100101B' to the upper byte of BTCON.

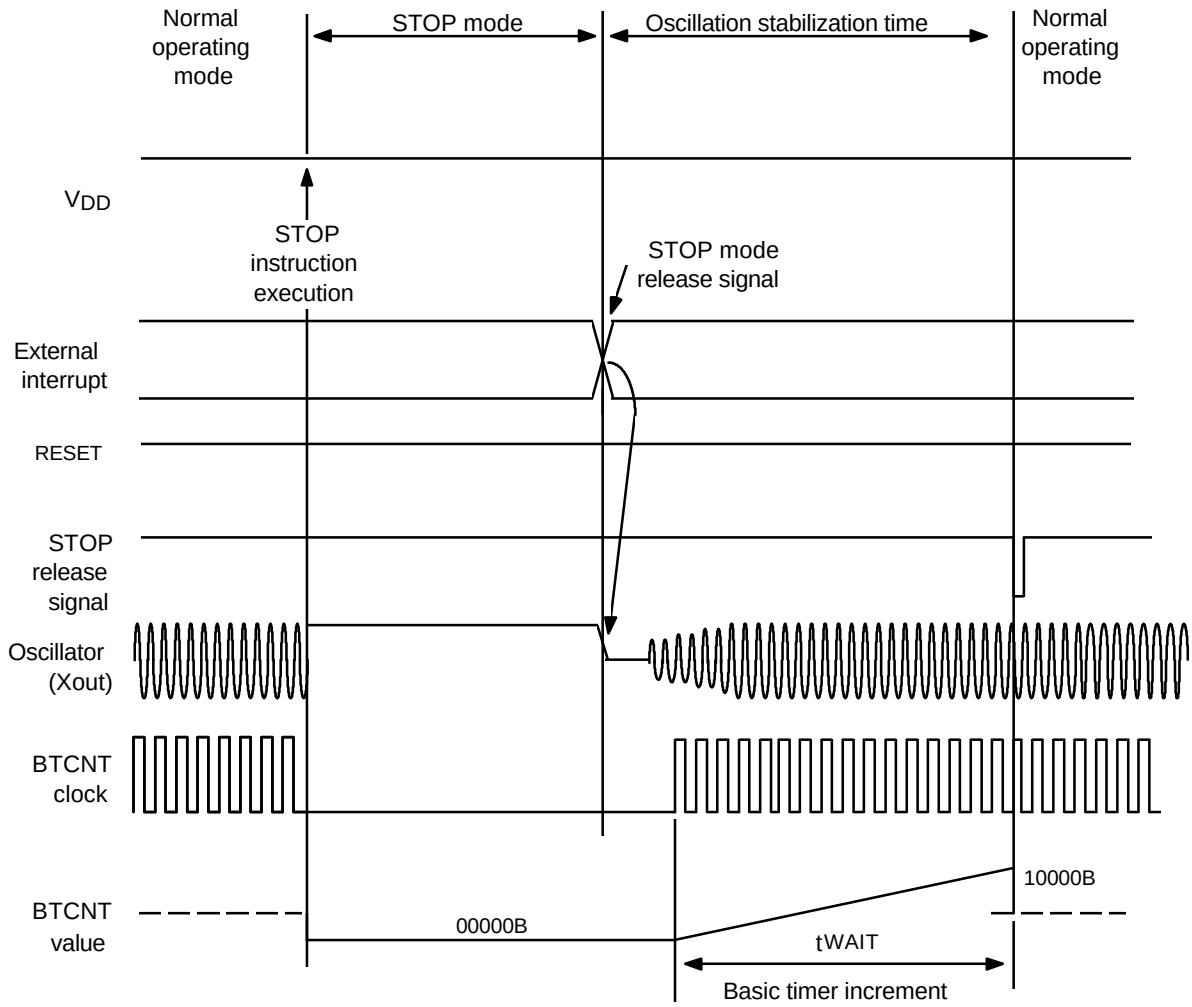
Table 12-14. System Operating Mode

Mode	Test Pin Condition		Each Port Setting on Reset
	Test 2	Test 2	
Normal 8-bit boot mode	V_{SS}	V_{SS}	Address bus: A0-A15 Data bus: D0-D7 Control bus: nRD, nWBE0 P1, P2, and P3.1: input with pull-down resistor P3.0 and P3.2-3.7: input with pull-up resistor Other port: input without pull-up or pull-down resistor.
Normal 16-bit boot mode	V_{SS}	V_{DD}	Address bus: A0-A15 Data bus: D0-D7 and D8-D15 Control bus: nRD, nWBE0 P1 and P3.1: input with pull-down resistor P3.0 and P3.2-3.7: input with pull-up resistor Other port: input without pull-up or pull-down resistor.
MDS 8-bit boot mode	V_{DD}	V_{SS}	Address bus: A0-A15 Data bus: D0-D7 Control bus: nRD, nWBE0 P1, P2, and P3.1: input with pull-down resistor P3.0 and P3.2-3.7: input with pull-up resistor P0: JTAG port Other port: input without pull-up or pull-down resistor.
MDS 16-bit boot mode	V_{DD}	V_{DD}	Address bus: A0-A15 Data bus: D0-D7 and D8-D15 Control bus: nRD, nWBE0 P1 and P3.1: input with pull-down resistor P3.0 and P3.2-3.7: input with pull-up resistor P0: JTAG port Other port: input without pull-up or pull-down resistor.



NOTE: Duration of the oscillator stabilization wait time, t_{WAIT}, when it is released by a Power-on-reset is 8192 x 16/fosc.
trst ≈ RC (R is external resister and C is on chip capacitor).

Figure 12-32. Oscillation Stabilization Time on RESET



NOTE: Duration of the oscillator stabilization wait time, tWAIT, it is released by an interrupt is determined by the setting in basic timer control register, BTCON.

BTCON.3	BTCON.2	tWAIT	tWAIT (When fosc is 20 MHz)
0	0	8192 x 16 x 1/ fosc	6.5536 ms
0	1	4096 x 16 x 1/ fosc	3.2768 ms
1	0	2048 x 16 x 1/ fosc	1.6384 ms
1	1	512 x 16 x 1/ fosc	0.4096 ms

Figure 12-33. Oscillation Stabilization Time on STOP Mode Release

POWER-DOWN MODES

STOP MODE

Stop mode is invoked by the setting SYSCON register bit 0 to high. In Stop mode, the operation of the CPU and all peripherals is halted. That is, the on-chip main oscillator stops and the supply current is reduced to less than 1 µA. All system functions stop when the clock "freezes," but data stored in the internal register file is retained. Stop mode can be released in one of two ways: by a reset or by an external interrupt (with RC delay).

NOTE

Do not use stop mode if you are using an external clock source because X_{IN} input must be restricted internally to V_{SS} to reduce current leakage.

Using RESET to Release Stop Mode

Stop mode is released when the RESET signal is released and returns to high level: all system and peripheral control registers are reset to their default hardware values and the contents of all data registers are retained. A reset operation automatically selects a slow clock (1/16) because CLKCON.3, CLKCON.4, and CLKCON.5 are cleared to '00B'. After the programmed oscillation stabilization interval has elapsed, the CPU starts the system initialization routine by fetching the program instruction stored in location 00 0000H.

Using an External Interrupt to Release Stop Mode

Only external interrupts with an RC-delay noise filter circuit can be used to release Stop mode. Which interrupt you can use to release Stop mode in a given situation depends on the microcontroller's current internal operating mode. The external interrupts in the KS17C4000 interrupt structure that can be used to release Stop mode are:

— External interrupts P6.7 (INT0), P4.5 (INT1), P1.0-P1.7 (INT2-INT9), and P7.0-P7.2 (INT10-INT12)

Please note the following conditions for Stop mode release:

— If you release Stop mode using an external interrupt, the current values in system and peripheral control registers are unchanged.

— If you use an external interrupt for Stop mode release, you can also program the duration of the oscillation stabilization interval. To do this, you must make the appropriate control and clock settings *before* entering Stop mode.

— When the Stop mode is released by external interrupt, the SYSCON.5, SYSCON.4, and SYSCON.3 bit-pair setting remains unchanged and the currently selected clock value is used.

— The external interrupt is serviced when the Stop mode release occurs. Following the service routine, the instruction immediately following the one that initiated Stop mode is executed.

IDLE MODE

Idle mode is invoked by the setting SYSCON register bit 1 to high. In Idle mode, CPU operations are halted while some peripherals remain active. During Idle mode, the internal clock signal is gated away from the CPU, but all peripherals timers remain active. Port pins retain the mode (input or output) they had at the time Idle mode was entered.

There are two ways to release Idle mode:

1. Execute a reset. All system and peripheral control registers are reset to their default values and the contents of all data registers are retained. The reset automatically selects a *slow clock (1/16)* because SYSCON.5, SYSCON.4, and SYSCON.3 are cleared to '000B'. If interrupts are masked, a reset is the only way to release Idle mode.
2. Activate any enabled interrupt, causing Idle mode to be released. When you use an interrupt to release Idle mode, the SYSCON.5, SYSCON.4, and SYSCON.3 register values remain unchanged, and the *currently selected clock* value is used. The interrupt is then serviced. When the return-from-interrupt (IRET) occurs, the instruction immediately following the one that initiated Idle mode is executed.

☞ PROGRAMMING TIP - Sample KS17C4000 Power Down Routine

The following sample program suggests programming method for the KS17C4000 power down mode:

```
LDR     R0, = 0x1ff0000 + 0xd003    ;  SYSCON register address
LDRB    R1, [R0]
ORRB    R1, R1, #1
STRB    R1, [R0]                    ;  Enter the stop mode
NOP                                 ;  Please insert the minimum 4 NOP instructions
NOP
NOP
NOP
NOP
•
•
```

; Before entering the idle mode, interrupt source, IntMask, and global interrupt flag are enabled, if necessary.

```
LDR     R0, =0x1ff0000 + 0xd003     ;  SYSCON register address
LDRB    R1, [R0]
ORRB    R1, R1, #2
STRB    R1, [R0}                    ;  Enter this idle mode
NOP                                 ;  Please insert the minimum 4 NOP instructions
NOP
NOP
NOP
NOP
•
•
```

13 INTERRUPT

OVERVIEW

The KS17C4000 interrupt structure has a total of 32 interrupt sources. Interrupt requests can be generated at internal function blocks or external pins. The ARM7TDMI core recognizes two kinds of interrupts: a normal interrupt request (IRQ), and a fast interrupt request (FIQ). Therefore, all KS17C4000 interrupts can be categorized as either IRQ or FIQ. The KS17C4000 interrupt controller extends the number of multiple interrupt sources that can be serviced by using three special registers, IntMode, IntPend, and IntMask:

— Interrupt mode register. Defines the interrupt mode, IRQ or FIQ, for each interrupt source.

— Interrupt pending register. Indicates that an interrupt request is pending (that is, when the I-flag or F-flag is set in the program status register, PSR). This status prevents any additional interrupt from being acknowledged. When a pending bit is set, the interrupt service routine starts whenever the I-flag or F-flag is cleared to 0. The service routine must clear the pending condition by writing a 1 to the appropriate pending bit.

— Interrupt mask register. Indicates that the current interrupt has been disabled as the corresponding mask bit is 0. If an interrupt mask bit is 1, the interrupt will be serviced normally.

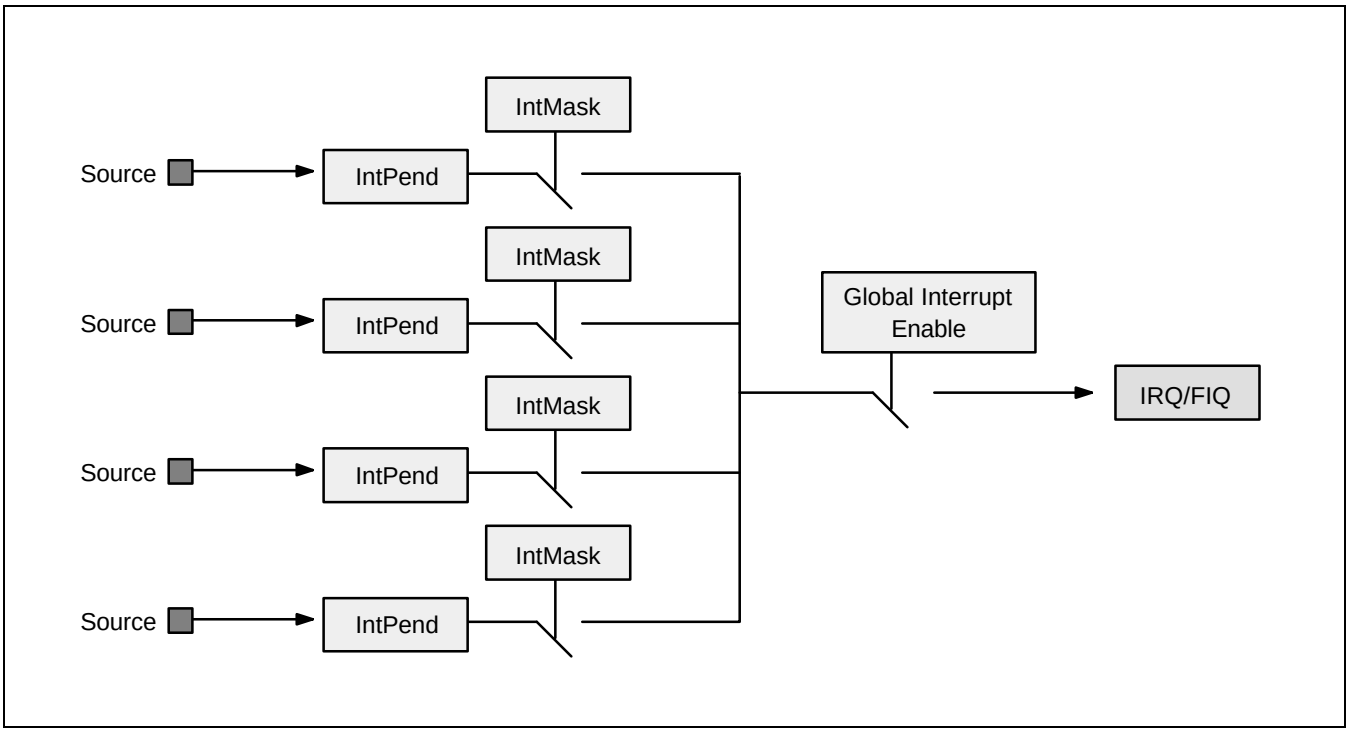


Figure 13-1. KS17C4000 Interrupt Structure

INTERRUPT SOURCES

The 32 interrupt sources in the KS17C4000 interrupt structure are describe, in brief, as follows:

[31] Interrupt 12 (P7.2) external interrupt
[30] Interrupt 11 (P7.1) external interrupt
[29] Interrupt 10 (P7.0) external interrupt
[28] A/D Conversion interrupt
[27] Interrupt 9 (P1.7) external interrupt
[26] Interrupt 8 (P1.6) external interrupt
[25] Interrupt 7 (P1.5) external interrupt
[24] Interrupt 6 (P1.4) external interrupt
[23] Interrupt 5 (P1.3) external interrupt
[22] Interrupt 4 (P1.2) external interrupt
[21] Interrupt 3 (P1.1) external interrupt
[20] Interrupt 2 (P1.0) external interrupt
[19] Serial I/O 1 interrupt
[18] Serial I/O 0 Interrupt
[17] Basic timer interrupt
[16] Timer 4 match/capture interrupt
[15] Timer 4 overflow interrupt
[14] Timer 3 match/capture interrupt
[13] Timer 3 overflow interrupt
[12] Timer 2 match/capture interrupt
[11] Timer 2 overflow interrupt
[10] Timer 1 match/capture interrupt
[9] Timer 1 overflow interrupt
[8] Timer 0 match/capture interrupt
[7] Timer 0 overflow interrupt
[6] DMA 1 interrupt
[5] DMA 0 interrupt
[4] UART error
[3] UART transmit interrupt
[2] UART receive interrupt
[1] Interrupt 1 (P4.5) external interrupt
[0] Interrupt 0 (P6.7) external interrupt

INTERRUPT CONTROLLER SPECIAL REGISTERS

INTERRUPT MODE REGISTER

Bits in the interrupt mode register, IntMode, specify if an interrupt is to be serviced as a fast or a normal interrupt. Each of the interrupt mode register, IntMode, corresponds to an interrupt source. When the source interrupt mode bit is set to 1, the interrupt is processed by the ARM7TDMI core in the FIQ (fast interrupt) mode. Otherwise, it is processed in the IRQ mode (normal interrupt).

Register	Offset Address	R/W	Description	Reset Value
IntMode	0xc000	R/W	Interrupt mode register	0000 0000h

[31] Interrupt 12 (P7.2) external interrupt
[30] Interrupt 11 (P7.1) external interrupt
[29] Interrupt 10 (P7.0) external interrupt
[28] A/D Conversion interrupt
[27] Interrupt 9 (P1.7) external interrupt
[26] Interrupt 8 (P1.6) external interrupt
[25] Interrupt 7 (P1.5) external interrupt
[24] Interrupt 6 (P1.4) external interrupt
[23] Interrupt 5 (P1.3) external interrupt
[22] Interrupt 4 (P1.2) external interrupt
[21] Interrupt 3 (P1.1) external interrupt
[20] Interrupt 2 (P1.0) external interrupt
[19] Serial I/O 1 interrupt
[18] Serial I/O 0 interrupt
[17] Basic timer interrupt
[16] Timer 4 match/capture interrupt
[15] Timer 4 overflow interrupt
[14] Timer 3 match/capture interrupt
[13] Timer 3 overflow interrupt
[12] Timer 2 match/capture interrupt
[11] Timer 2 overflow interrupt
[10] Timer 1 match/capture interrupt
[9] Timer 1 overflow interrupt
[8] Timer 0 match/capture interrupt
[7] Timer 0 overflow interrupt
[6] DMA 1 interrupt
[5] DMA 0 interrupt
[4] UART error
[3] UART transmit interrupt
[2] UART receive interrupt
[1] Interrupt 1 (P4.5) external interrupt
[0] Interrupt 0 (P6.7) external interrupt

INTERRUPT PENDING REGISTER

The interrupt pending register, IntPend, contains interrupt pending bits for each interrupt source. Each of the interrupt pending register, IntPend, corresponds to an interrupt source. When an interrupt request is generated, it will be masked by the CPU, if the I-flag or F-flag in the corresponding status register (PSR) is set. In this case, the source interrupt pending bit is set to 1. When the I-flag or F-flag (IRQ, FIQ) is cleared in the PSR, the interrupt routine is initialized. The service routine must then clear the appropriate pending bit.

Register	Offset Address	R/W	Description	Reset Value
IntPend	0xc004	R/W	Interrupt pending register	0000 0000h

[31] Interrupt 12 (P7.2) external interrupt
[30] Interrupt 11 (P7.1) external interrupt
[29] Interrupt 10 (P7.0) external interrupt
[28] A/D Conversion interrupt
[27] Interrupt 9 (P1.7) external interrupt
[26] Interrupt 8 (P1.6) external interrupt
[25] Interrupt 7 (P1.5) external interrupt
[24] Interrupt 6 (P1.4) external interrupt
[23] Interrupt 5 (P1.3) external interrupt
[22] Interrupt 4 (P1.2) external interrupt
[21] Interrupt 3 (P1.1) external interrupt
[20] Interrupt 2 (P1.0) external interrupt
[19] Serial I/O 1 interrupt
[18] Serial I/O 0 interrupt
[17] Basic timer interrupt
[16] Timer 4 match/capture interrupt
[15] Timer 4 overflow interrupt
[14] Timer 3 match/cCapture interrupt
[13] Timer 3 overflow interrupt
[12] Timer 2 match/capture interrupt
[11] Timer 2 overflow interrupt
[10] Timer 1 match/capture interrupt
[9] Timer 1 overflow interrupt
[8] Timer 0 match/capture interrupt
[7] Timer 0 overflow interrupt
[6] DMA 1 interrupt
[5] DMA 0 interrupt
[4] UART error
[3] UART transmit interrupt
[2] UART receive interrupt
[1] Interrupt 1 (P4.5) external interrupt
[0] Interrupt 0 (P6.7) external interrupt

INTERRUPT MASK REGISTER

The interrupt mask register, IntMask, contains interrupt mask bits for each interrupt source. Each of the interrupt mask register, IntMask, corresponds to an interrupt source. When an interrupt source mask bit is 0, the interrupt is not serviced by the CPU when the corresponding interrupt request is generated. If the mask bit is 1, the interrupt is serviced upon request.

Register	Offset Address	R/W	Description	Reset Value
IntMask	0xc008	R/W	Interrupt mask register	0000 0000h

[31] Interrupt 12(P7.2) external interrupt
[30] Interrupt 11(P7.1) external interrupt
[29] Interrupt 10(P7.0) external interrupt
[28] A/D Conversion interrupt.
[27] Interrupt 9(P1.7) external interrupt
[26] Interrupt 8(P1.6) external interrupt
[25] Interrupt 7(P1.5) external interrupt.
[24] Interrupt 6(P1.4) external interrupt
[23] Interrupt 5(P1.3) external interrupt.
[22] Interrupt 4(P1.2) external interrupt.
[21] Interrupt 3(P1.1) external interrupt.
[20] Interrupt 2(P1.0) external interrupt
[19] Serial I/O 1 interrupt
[18] Serial I/O 0 Interrupt
[17] Basic Timer Interrupt.
[16] Timer 4 Match/Capture interrupt.
[15] Timer 4 Overflow interrupt.
[14] Timer 3 Match/Capture interrupt.
[13] Timer 3 Overflow interrupt.
[12] Timer 2 Match/Capture interrupt.
[11] Timer 2 Overflow interrupt.
[10] Timer 1 Match/Capture interrupt.
[9] Timer 1 Overflow interrupt.
[8] Timer 0 Match/Capture interrupt.
[7] Timer 0 Overflow interrupt.
[6] DMA 1 interrupt.
[5] DMA 0 interrupt.
[4] UART error.
[3] UART transmit interrupt.
[2] UART receive interrupt.
[1] Interrupt 1(P4.5) external interrupt.
[0] Interrupt 0(P6.7) external interrupt.

NOTES

14 SPECIAL FUNCTION REGISTERS

This chapter describes the KS17C4000 Special Function Registers. The SFR block has an SRAM area for stack or data memory and special registers that control peripheral blocks.

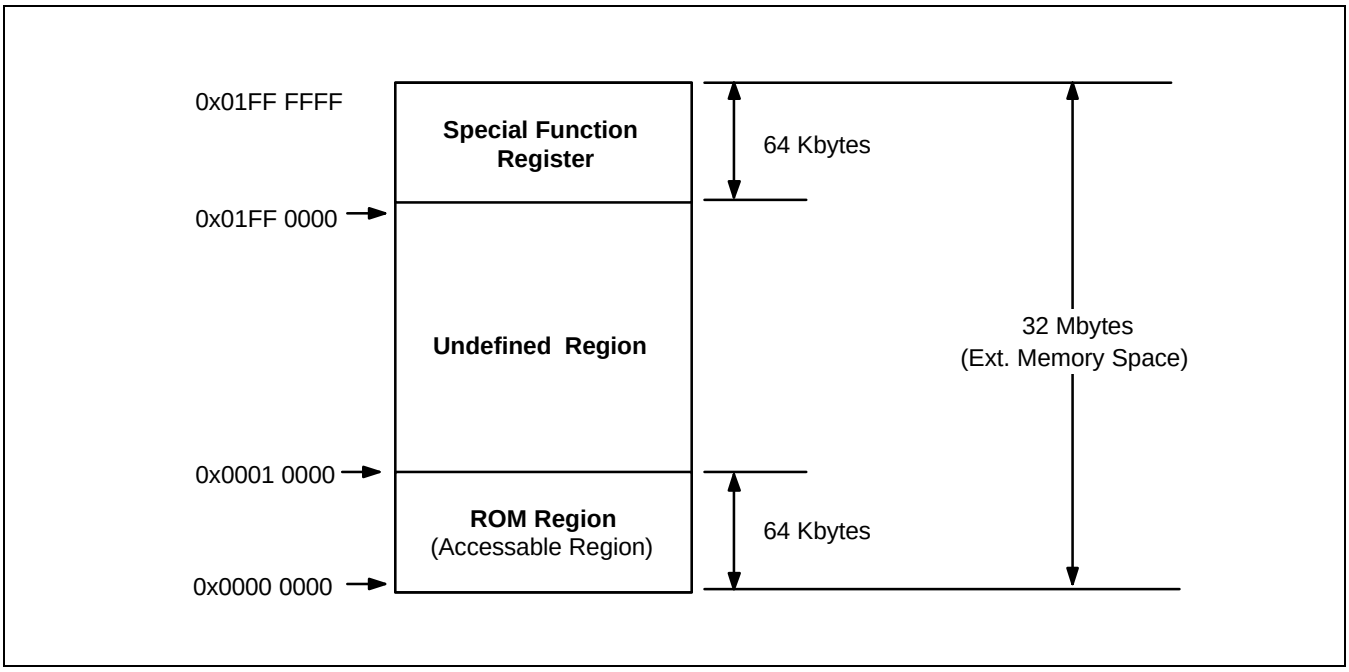


Figure 14-1. KS17C4000 Memory Map (Default Map after Reset)

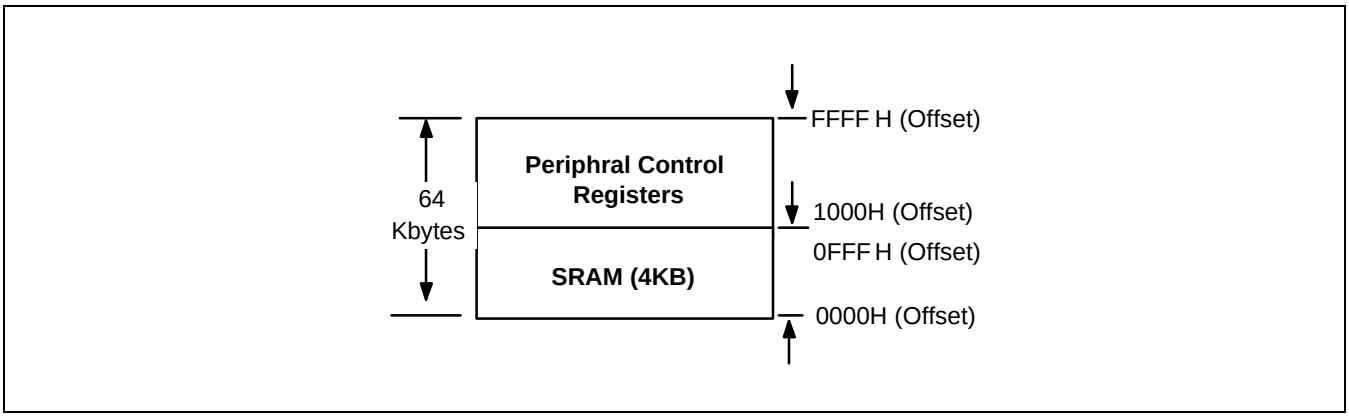


Figure 14-2. Special Function Register

KS17C4000 SPECIAL REGISTERS

Table 14-1. Special Registers

Group	Registers	Offset	R/W	Description	Access	Reset Value
System Manager	SYSCFG	0x1000	R/W	System Configuration register	W	3ff1h
	MEMCON0	0x2000	R/W	Memory Bank 0 control register	W	0080 0600h
	MEMCON1	0x2004	R/W	Memory Bank 1 control register	W	0000 0000h
	MEMCON2	0x2008	R/W	Memory Bank 2 control register	W	0000 0000h
	MEMCON3	0x200c	R/W	Memory Bank 3 control register	W	0000 0000h
	REFCON	0x2010	R/W	DRAM refresh control register	W	0 0001h
	EDVCON	0x2014	R/W	Extra device control register	W	0000h
	EXTPORT	0x201a	R/W	External Port Buffer register	B/H	0000h
	ECSDATA0	0x201c	R/W	Extra Chip Selection 0 Data register	B/H	0000h
	ECSDATA1	0x201e	R/W	Extra Chip Selection 1 Data register	B/H	0000h
DMA	DMA0CON	0x3000	R/W	DMA 0 control register	W	0000h
	DMA0SRC	0x3004	R/W	DMA 0 source address register	W	000 0000h
	DMA0DST	0x3008	R	DMA 0 destination address register	W	000 0000h
	DMA0CNT	0x300c	R/W	DMA 0 transfer count register	W	00 0000h
	DMA1CON	0x4000	R/W	DMA 1 control register	W	0000h
	DMA1SRC	0x4004	R/W	DMA 1 source address register	W	000 0000h
	DMA1DST	0x4008	R	DMA 1 destination address register	W	000 0000h
	DMA1CNT	0x400c	R/W	DMA 1 transfer count register	W	00 0000h
UART	LCON	0x5003	R/W	UART line control register	B	00h
	UCON	0x5007	R/W	UART control register	B	00h
	USSR	0x500b	R/W	UART status register	B	c0h
	TBR	0x500f	W	UART transmit buffer control register	B	xxh
	RBR	0x5013	R	UART receive buffer control register	B	xxh
	UBRDR	0x5016	R/W	UART baud rate divisor register	H	0000h
	BRDCNT	0x501a	–	Baud counter for test	H	–
	BRDCLK	0x501f	–	Baud clock for test	B	–

Table 14-1. Special Registers (Continued)

Group	Registers	Offset	R/W	Description	Access	Reset Value
SIO	SBIDR0	0x6000	R/W	Serial I/O port 0 byte interval data reg	B	00h
	SBRDR0	0x6001	R/W	Serial I/O port 0 baud rate divisor reg	B	00h
	SIO0DATA	0x6002	R/W	Serial I/O port 0 data register	B	00h
	SIO0CON	0x6003	R/W	Serial I/O port 0 control register	B	00h
	SBIDR1	0x7000	R/W	Serial I/O port 1 byte interval data reg	B	00h
	SBRDR1	0x7001	R/W	Serial I/O port 1 baud rate divisor reg	B	00h
	SIO1DATA	0x7002	R/W	Serial I/O port 1 data register	B	00h
	SIO1CON	0x7003	R/W	Serial I/O port 1 control register	B	00h
A/D C	ADCON	0x8003	R/W	A/D C control register	B only	08h
	ADDATA	0x8007	R	A/D Conversion result data register	B only	xxh
Timer 0	T0DATA	0x9000	R/W	Timer 0 data register	H	ffffh
	T0PRE	0x9002	R/W	Timer 0 prescaler register	B	ffh
	T0CON	0x9003	R/W	Timer 0 control register	B	00h
	T0CNT	0x9006	R	Timer 0 counter register	H	0000h
Timer 1	T1DATA	0x9010	R/W	Timer 1 data register	H	ffffh
	T1PRE	0x9012	R/W	Timer 1 prescaler register	B	ffh
	T1CON	0x9013	R/W	Timer 1 control register	B	00h
	T1CNT	0x9016	R	Timer 1 counter register	H	0000h
Timer 2	T2DATA	0x9020	R/W	Timer 2 data register	H	ffffh
	T2PRE	0x9022	R/W	Timer 2 prescaler register	B	ffh
	T2CON	0x9023	R/W	Timer 2 control register	B	00h
	T2CNT	0x9026	R	Timer 2 counter register	H	0000h
Timer 3	T3DATA	0x9031	R/W	Timer 3 data register	B	ffh
	T3PRE	0x9032	R/W	Timer 3 prescaler register	B	ffh
	T3CON	0x9033	R/W	Timer 3 control register	B	00h
	T3CNT	0x9037	R	Timer 3 counter register	B	00h
Timer 4	T4DATA	0x9041	R/W	Timer 4 data register	B	ffh
	T4PRE	0x9042	R/W	Timer 4 prescaler register	B	ffh
	T4CON	0x9043	R/W	Timer 4 control register	B	00h
	T4CNT	0x9047	R	Timer 4 counter register	B	00h
BT & WDT	BTCON	0xa002	R/W	Basic timer control register	H/B	0000h
	BTCNT	0xa007	R	Basic timer counter register	B	00h

Table 14-1. Special Registers (Continued)

Group	Registers	Offset	R/W	Description	Access	Reset Value
I/O Port	Port 0	0xb000	R/W	Port 0 data register	B	00h
	Port 1	0xb001	R/W	Port 1 data register	B	00h
	Port 2	0xb002	R/W	Port 2 data register	B	00h
	Port 3	0xb003	R/W	Port 3 data register	B	00h
	Port 4	0xb004	R/W	Port 4 data register	B	00h
	Port 5	0xb005	R/W	Port 5 data register	B	00h
	Port 6	0xb006	R/W	Port 6 data register	B	00h
	Port 7	0xb007	R	Port 7 data register	B	xxh
	P0BR	0xb008	R/W	Port 0 buffer reg (for real time output)	B	00h
	EINTCON	0xb00a	R/W	External Interrupt Control register	H	0000h
	EINTMOD	0xb00c	R/W	External Interrupt Mode register	W	000 0000h
I/O Port Control Register	P0CON	0xb010	R/W	Port 0 control register	H	000h
	P1CON	0xb012	R/W	Port 1 control register	H	0000h
	P2CON	0xb014	R/W	Port 2 control register	H	0000h
	P3CON	0xb018	R/W	Port 3 control register	H	0000h
	P4CON	0xb01a	R/W	Port 4 control register	H	000h
	P5CON	0xb01c	R/W	Port 5 control register	W	00 0000h
	P6CON	0xb020	R/W	Port 6 control register	H	000h
	P7CON	0xb022	R/W	Port 7 control register	B	00h
I/O Port Resistor Control Register	P0PUR	0xb028	R/W	Port 0 pull-up control register	B	00h
	P1PDR	0xb029	R/W	Port 1 pull-down control register	B	FFh
	P2PDR	0xb02a	R/W	Port 2 pull-down control register	B	FFh
	P3PUR	0xb02b	R/W	Port 3 pull-up/down control register	B	FFh
	P4PUR	0xb02c	R/W	Port 4 pull-up control register	B	0h
	P5PUR	0xb02d	R/W	Port 5 pull-up control register	B	00h
	P6PUR	0xb02e	R/W	Port 6 pull-up control register	B	00h
	P7PUR	0xb02f	R/W	Port 7 pull-up control register	B	00h
Interrupt Contro-ller	INTMODE	0xc000	R/W	Interrupt Mode register	W	0000 0000h
	INTPEND	0xc004	R/W	Interrupt Pending register	W	0000 0000h
	INTMASK	0xc008	R/W	Interrupt Mask register	W	0000 0000h
Others	SYSCON	0xd003	R/W	System Control register	B	00h
SRAM	SRAM	0x0000	R/W	Internal SRAM area	B	xxh
		–	–	–	B	–
		0x0FFF	R/W	Internal SRAM area (4-K Bytes)	B	xxh

SYSTEM CONTROL REGISTER

The system control register, SYSCON, is used to control the system operation of the chip.

Register	Offset Address	R/W	Description	Reset Value
SYSCON	0xd003	R/W	System Control register	00h

[0]	STOP bit	This bit value determines whether stop mode is enabled or disabled.
[1]	IDLE bit	This bit value determines whether idle mode is enabled or disabled.
[5:3]	Clock selection bit	This bit value determines whether divide two is enabled or disabled. 000: Select MCLK/16, 001: Select MCLK/8 010: MCLK/2, 011: MCLK/1, 100: MCLK/1024
[6]	Global interrupt enable bit	This bit value controls enabling or disabling the interrupt.

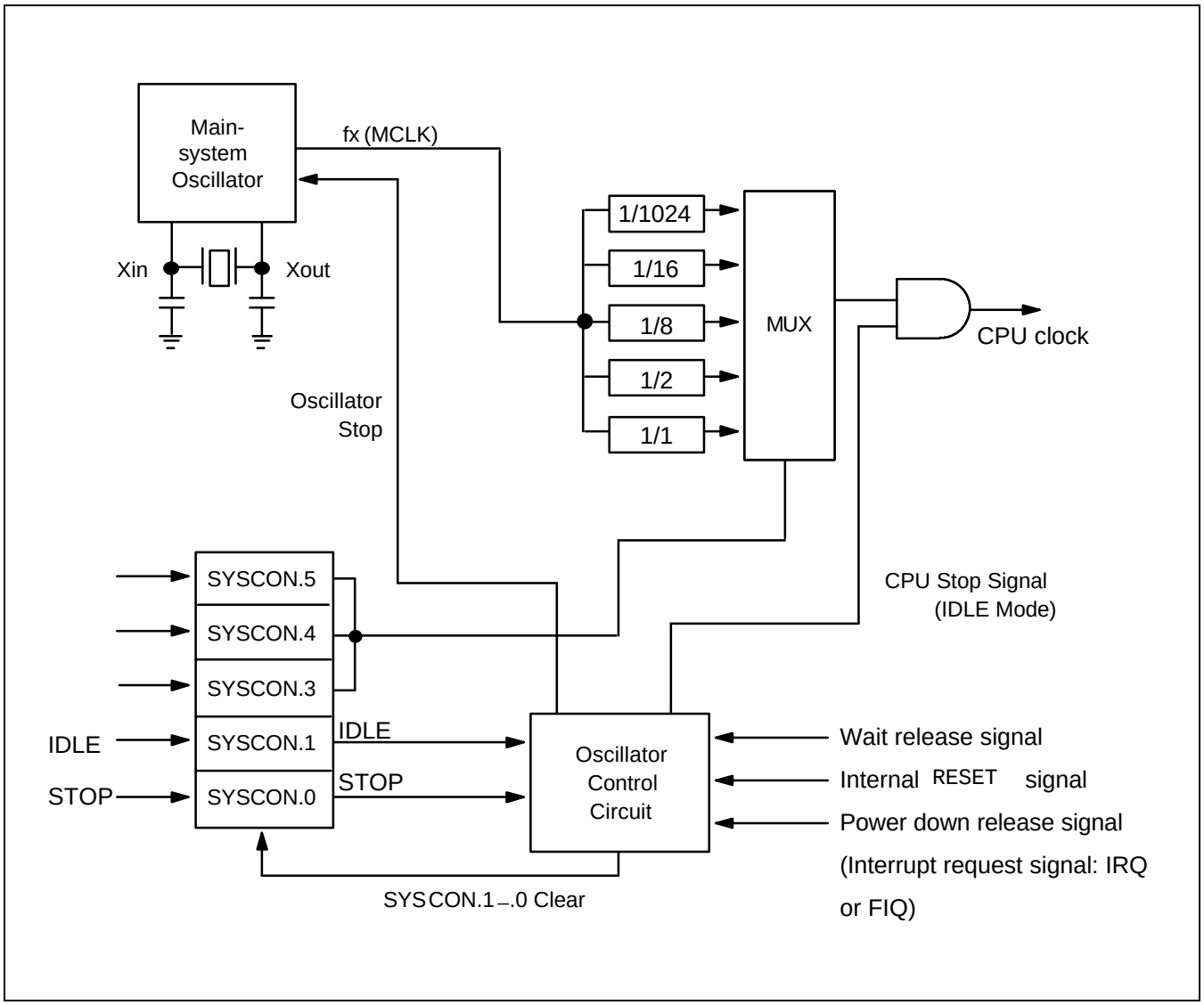


Figure 14-3. Clock Circuit Diagram

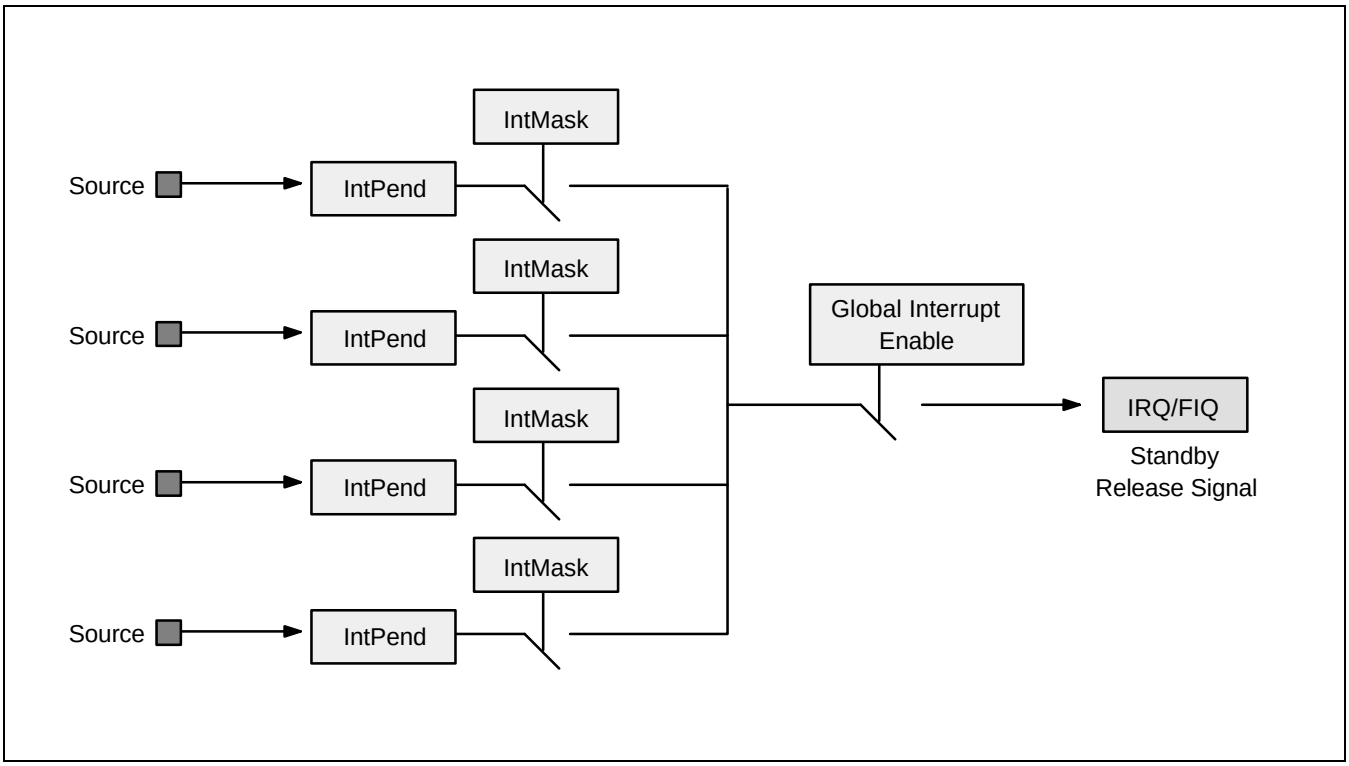


Figure 14-4. Interrupt Structure

NOTES

15 ELECTRICAL DATA

OVERVIEW

This chapter describes the KS17C4000 electrical data. Information is presented according to the following Table of Contents:

Table 15-1. Absolute Maximum Ratings

(T_A = 25°C)

Parameter	Symbol	Conditions	Rating	Unit
Supply voltage	V_{DD}		– 0.3 to + 6.5	V
Input voltage	V_I	All ports	– 0.3 to V_{DD} + 0.3	V
Output voltage	V_O		– 0.3 to V_{DD} + 0.3	V
Output current high	I_{OH}	One I/O pin active	– 15	mA
		All I/O pins active	– 100	
Output current low	I_{OL}	One I/O pin active	+ 20	mA
		Total pin current for ports 0, 1, 2, 3, 4, 5, 6	+ 150	
Operating temperature	T_A		– 40 to + 85	°C
Storage temperature	T_{STG}		– 65 to + 150	°C

Table 15-2. D.C. Electrical Characteristics

(T_A = – 40°C to + 85°C, V_{DD} = 2.7 V to 5.5 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Operating Voltage	V_{DD}	Fosc = 33 MHz, 128 pin	4.5	–	5.5	V
		Fosc = 20 MHz, 100 pin	4.5	–	5.5	
Input high voltage	V_{IH1}	Test1,2, Reset, P1, P4.5, P5, P6	0.8 V_{DD}	–	V_{DD}	V
	V_{IH2}	All input pins except V_{IH1} and V_{IH3}	0.7 V_{DD}			
	V_{IH3}	X_{IN}	V_{DD}– 0.5			
Input low voltage	V_{IL1}	Test1,2, Reset, P1, P4.5, P5, P6	0	–	0.2 V_{DD}	V
	V_{IL2}	All input pins except V_{IL1} and V_{IL3}			0.3 V_{DD}	
	V_{IL2}	X_{IN}			0.4	
Output high voltage	V_{OH1}	V_{DD} = 4.5 V to 5.5 V I_{OH} = – 2 mA	V_{DD}– 1.0	–	–	V
Output low voltage	V_{OL1}	V_{DD} = 4.5 V to 5.5 V I_{OL} = 2 mA All output pins except V_{OL2}	–	–	0.4	V
	V_{OL2}	V_{DD} = 4.5 V to 5.5 V I_{OL} = 8 mA A0-A15, D0-7, P2	–	–	0.4	
Input high leakage current	I_{LIH1}	V_{IN} = V_{DD} All input pins except X_{IN}	–	–	3	uA
	I_{LIH2}	V_{IN} = V_{DD} X_{IN}			20	
Input low leakage current	I_{LIL1}	V_{IN} = 0 V All input pins except X_{IN}, and RESET	–	–	– 3	uA
	I_{LIL2}	V_{IN} = 0 V X_{IN}			–- 20	
Output high leakage current	I_{LOH}	V_{OUT} = V_{DD} All output pins	–	–	5	uA
Output low leakage current	I_{LOL}	V_{OUT} = 0 V	–	–	– 5	uA

Table 15-2. D.C. Electrical Characteristics (Continued)

(T_A = - 40°C to + 85°C, V_{DD} = 2.7 V to 5.5 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Pull-up or pull-down resistor	R_{L1}	V_{IN} = 0 V or 5 V; V_{DD} = 5V Ports 0, 1, 2, 3, 4, 5, 6 and 7	30	50	70	KΩ
	R_{L2}	V_{IN} = 0 V; V_{DD} = 5V RESET only	30	50	70	
Oscillator Feedback resistor	R_{OSC}	V_{DD} = 5 V, T_A = +25°C X_{IN} = V_{DD}, X_{OUT} = 0V	600	1000	14000	
Supply current (note)	I_{DD1}	V_{DD} = 5 V 33 MHz crystal oscillator	–	98	150	mA
		V_{DD} = 5 V 20 MHz crystal oscillator		35	70	
	I_{DD2}	Idle mode: V_{DD}= 5V 33 MHz crystal oscillator		7.7	15	
		Idle mode: V_{DD} = 5V 20 MHz crystal oscillator		5.3	13	
	I_{DD3}	Stop mode; V_{DD} = 5 V		0.3	1	uA

NOTE: Supply current does not include current drawn through internal pull-up resistors or external output current loads.

Table 15-3. A.C. Electrical Characteristics

(T_A = - 40 °C to + 85 °C, V_{DD} = 2.7 V to 5.5 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Interrupt input high, low width	t_{INTH}, t_{INTL}	P4.5, P6.7, P1.0–P1.7, P7.0–P7.2 at V_{DD} = 5 V	200	–	–	nS
RESET input low width	t_{RSL}	V_{DD} = 5 V	128	–	–	MCLK

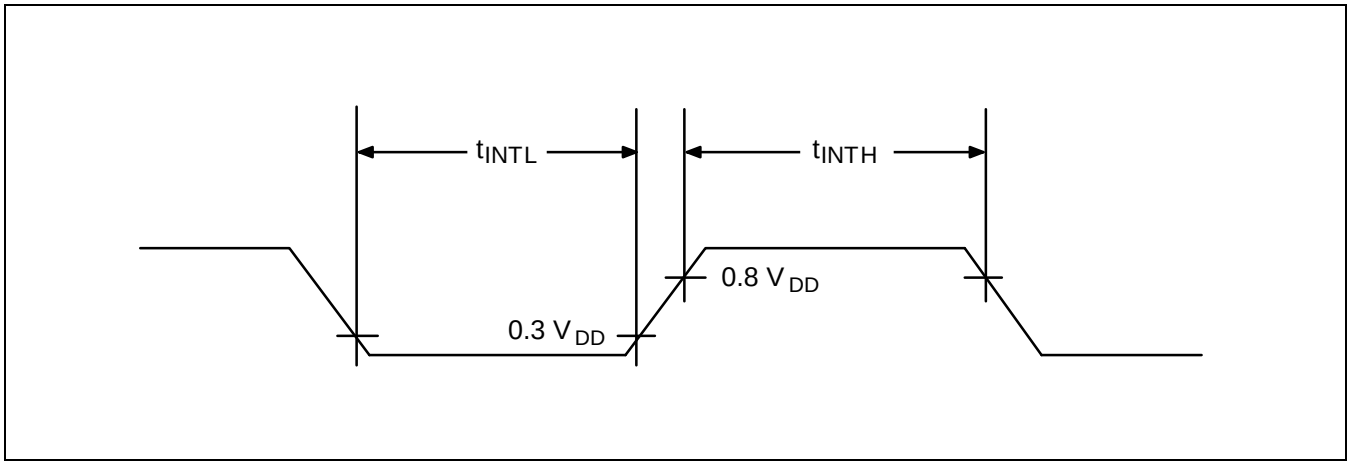


Figure 15-1. Input Timing for External Interrupts (Ports 4 and 5)

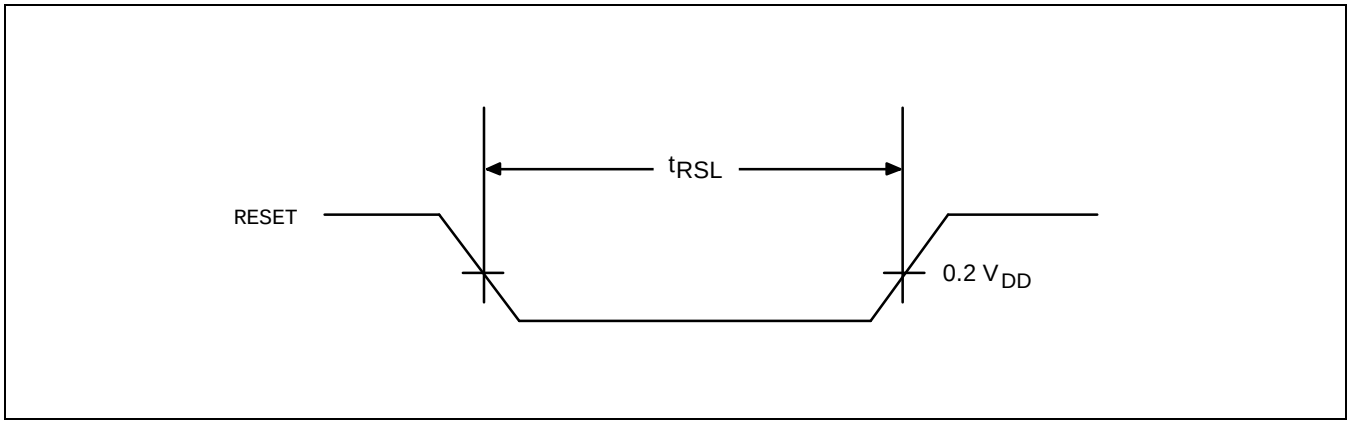


Figure 15-2. Input Timing for RESET

Table 15-4. Input/Output Capacitance

(T_A = – 40 °C to + 85 °C, V_{DD} = 0 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Input capacitance	C_{IN}	f = 1 MHz; unmeasured pins are returned to V_{SS}	–	–	10	pF
Output capacitance	C_{OUT}					
I/O capacitance	C_{IO}					

Table 15-5. Data Retention Supply Voltage in Stop Mode

(T_A = – 40 °C to + 85 °C)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Data retention supply voltage	V_{DDDR}		2	–	5.5	V
Data retention supply current	I_{DDDR}	V_{DDDR} = 2 V	–	–	1	uA

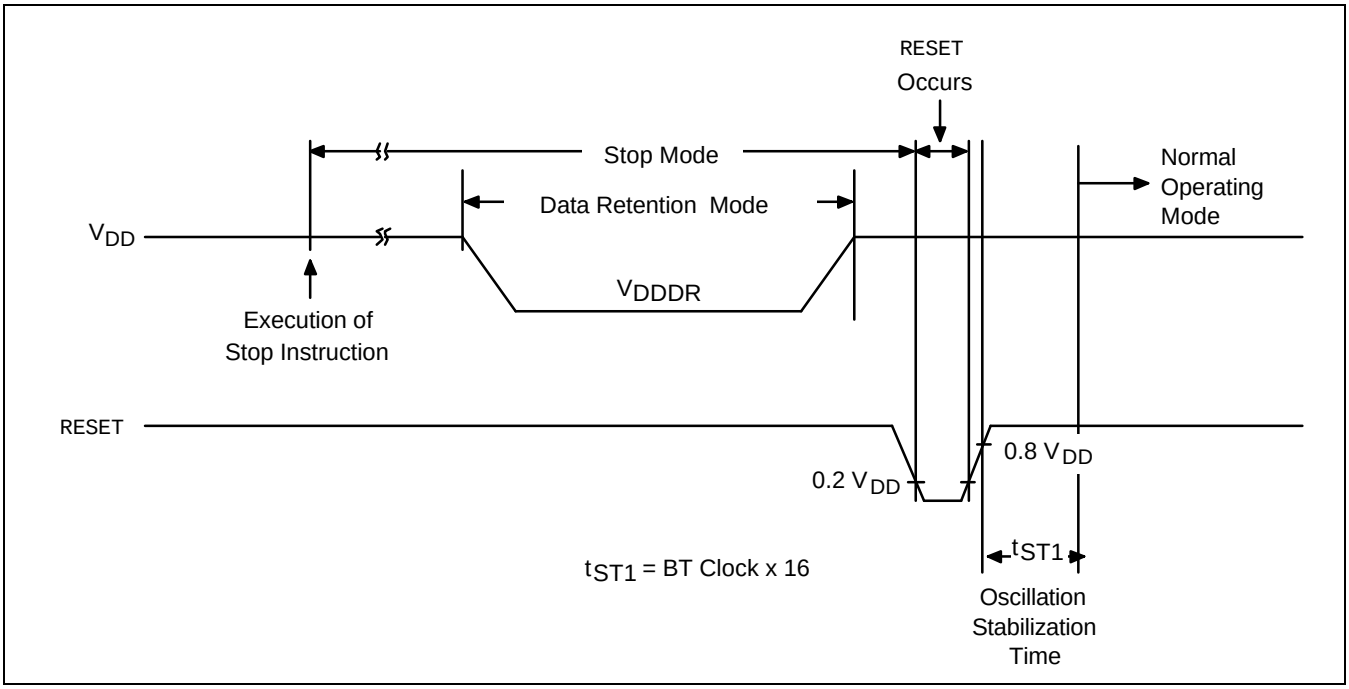


Figure 15-3. Stop Mode Release Timing Initiated by RESET

Table 15-6. A/D Converter Electrical Characteristics

(T_A = - 40 °C to + 85 °C, V_{DD} = 2.7 V to 5.5 V, V_{SS} = 0 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Resolution			–	8	–	bit
Total accuracy		V_{DD} = 5 V Conversion time = 5us	–	–	± 2	LSB
Integral Linearity Error	ILE	AV_{REF} = 5 V		–	± 1	
Differential Linearity Error	DLE	AV_{SS} = 0 V		–	± 1	
Offset Error of Top	EOT			± 1	± 2	
Offset Error of Bottom	EOB			± 0.5	± 2	
Conversion time (1)	t_{CON}	–	8	–	–	us
Analog input voltage	V_{IAN}	–	AV_{SS}	–	AV_{REF}	V
Analog input impedance	R_{AN}	–	2	1000	–	Mohm
Analog reference voltage	AV_{REF}	–	2.5	–	V_{DD}	V
Analog ground	AV_{SS}	–	V_{SS}	–	V_{SS}+ 0.3	V
Analog input current	I_{ADIN}	AV_{REF} = V_{DD} = 5 V	–	–	10	uA
Analog block current (2)	I_{ADC}	AV_{REF} = V_{DD} = 5 V		1	3	mA
		AV_{REF} = V_{DD} = 3 V		0.5	1.5	mA
		AV_{REF} = V_{DD} = 5 V When Power Down mode		100	500	nA

NOTES:
1. 'Conversion time' is the time required from the moment a conversion operation starts until it ends.
2. I_{ADC} is an operating current during A/D conversion.

Table 15-7. Serial Port Timing Characteristics in Mode 0 (10 MHz)

(T_A = - 40 °C to + 85 °C, V_{DD} = 4.5 V to 5.5 V, V_{SS} = 0 V)

Parameter	Symbol	Min	Typ	Max	Unit
Serial port clock cycle time	t_{SCK}	400	–	–	ns
Serial port clock high, low width	t_{HIGH}, t_{LOW}	190	–	–	

NOTE: All time units are in ns and assume a 10 MHz input frequency.

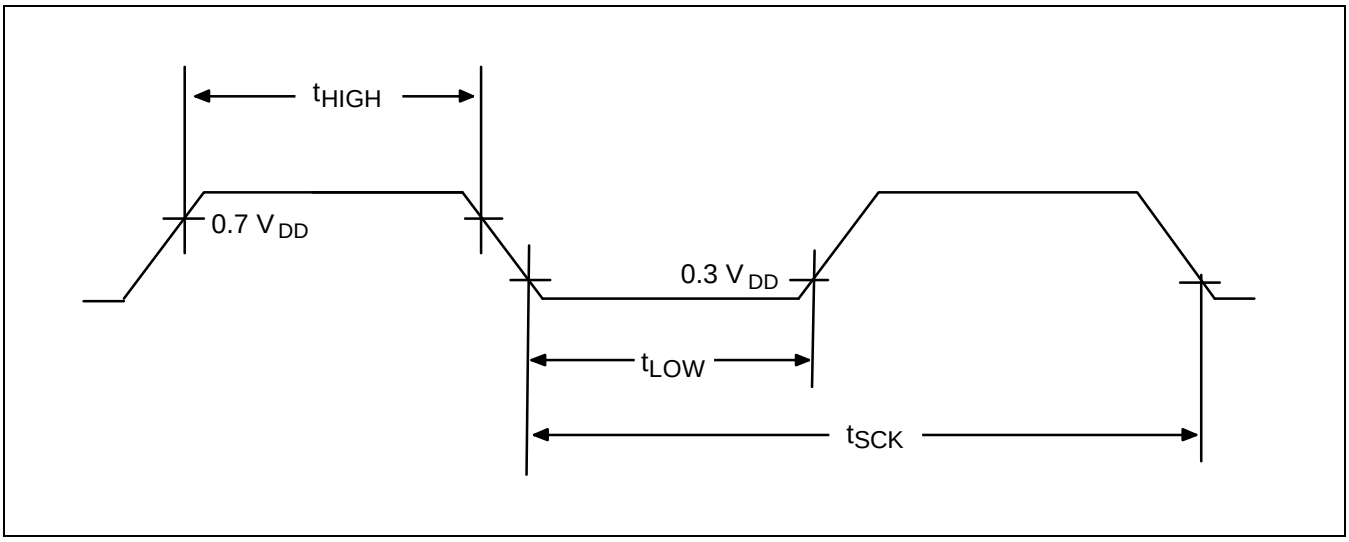


Figure 15-5. Serial Clock Waveform

Table 15-8. Main Oscillator Frequency (f_{OSC1})

(T_A = – 40 °C + 85 °C, V_{DD} = 2.7 V to 5.5 V)

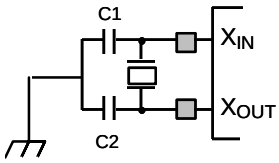
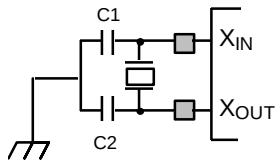
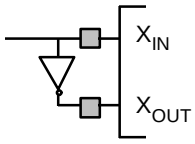
Oscillator	Clock Circuit	Test Condition	Min	Typ	Max	Unit
Crystal		CPU clock oscillation frequency	4	–	33	MHz
Ceramic		CPU clock oscillation frequency	4	–	33	MHz
External clock		X_{IN} input frequency	4	–	33	MHz

Table 15-9. Main Oscillator Clock Stabilization Time (t_{ST1})

(T_A = – 40 °C + 85 °C, V_{DD} = 2.7 V to 5.5 V)

Oscillator	Test Condition	Min	Typ	Max	Unit
Crystal	V_{DD} = 2.7 V to 5.5 V	–	–	20	ms
Ceramic	Stabilization occurs when V_{DD} is equal to the minimum oscillator voltage range.	–	–	10	ms
External clock	X_{IN} input high and low level width (t_{XH}, t_{XL})	15	–	125	ns

NOTE: Oscillation stabilization time (t_{ST1}) is the time required for the CPU clock to return to its normal oscillation frequency after a power-on occurs, or when Stop mode is ended by a RESET signal. The RESET should therefore be held at low level until the t_{ST1} time elapse (see Figure 15-3).

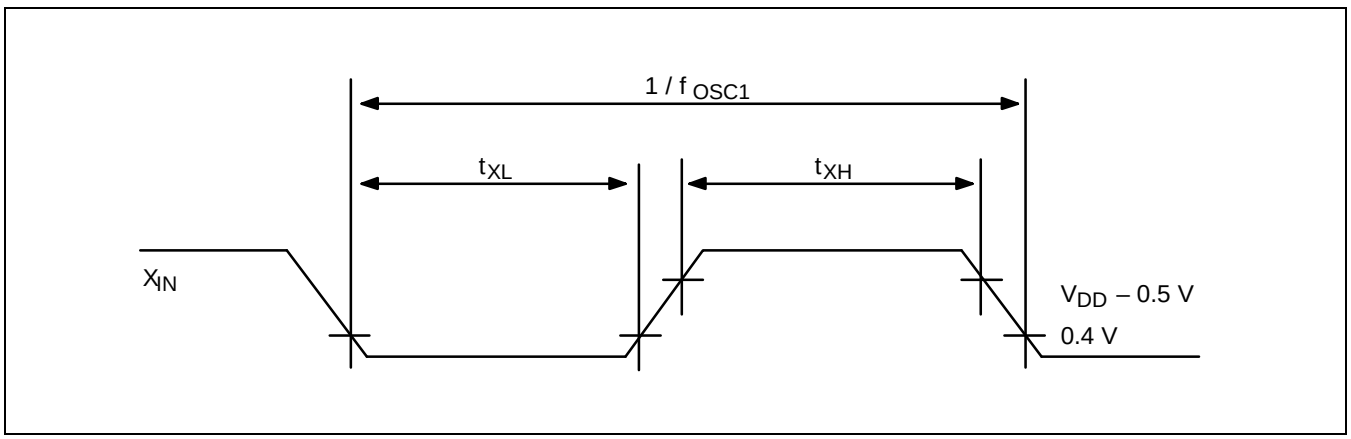


Figure 15-6. Clock Timing Measurement at X_{IN}

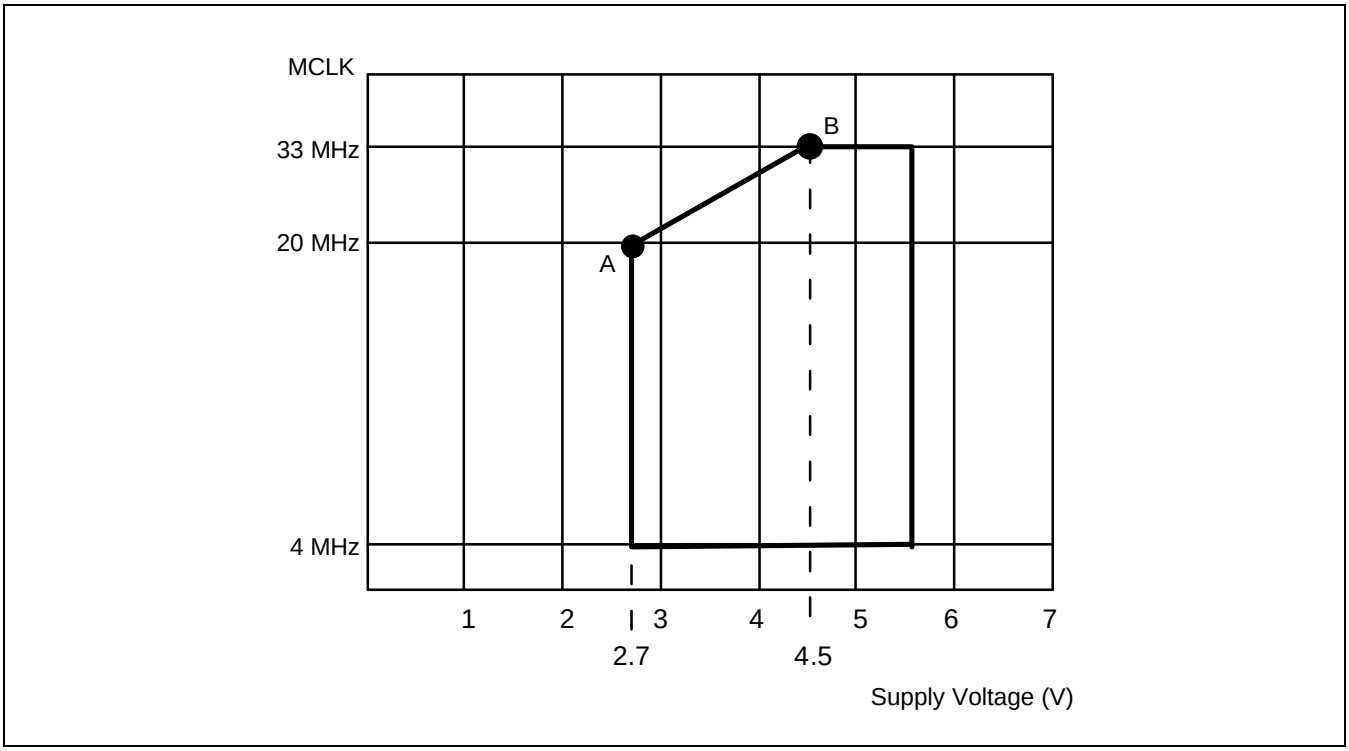


Figure 15-7. Operating Voltage Range

NOTES

16 MECHANICAL DATA

OVERVIEW

The KS17C4000 microcontroller is currently available in 100-pin QFP(100-QFP-1420C) and TQFP(100-TQFP-1414) package.

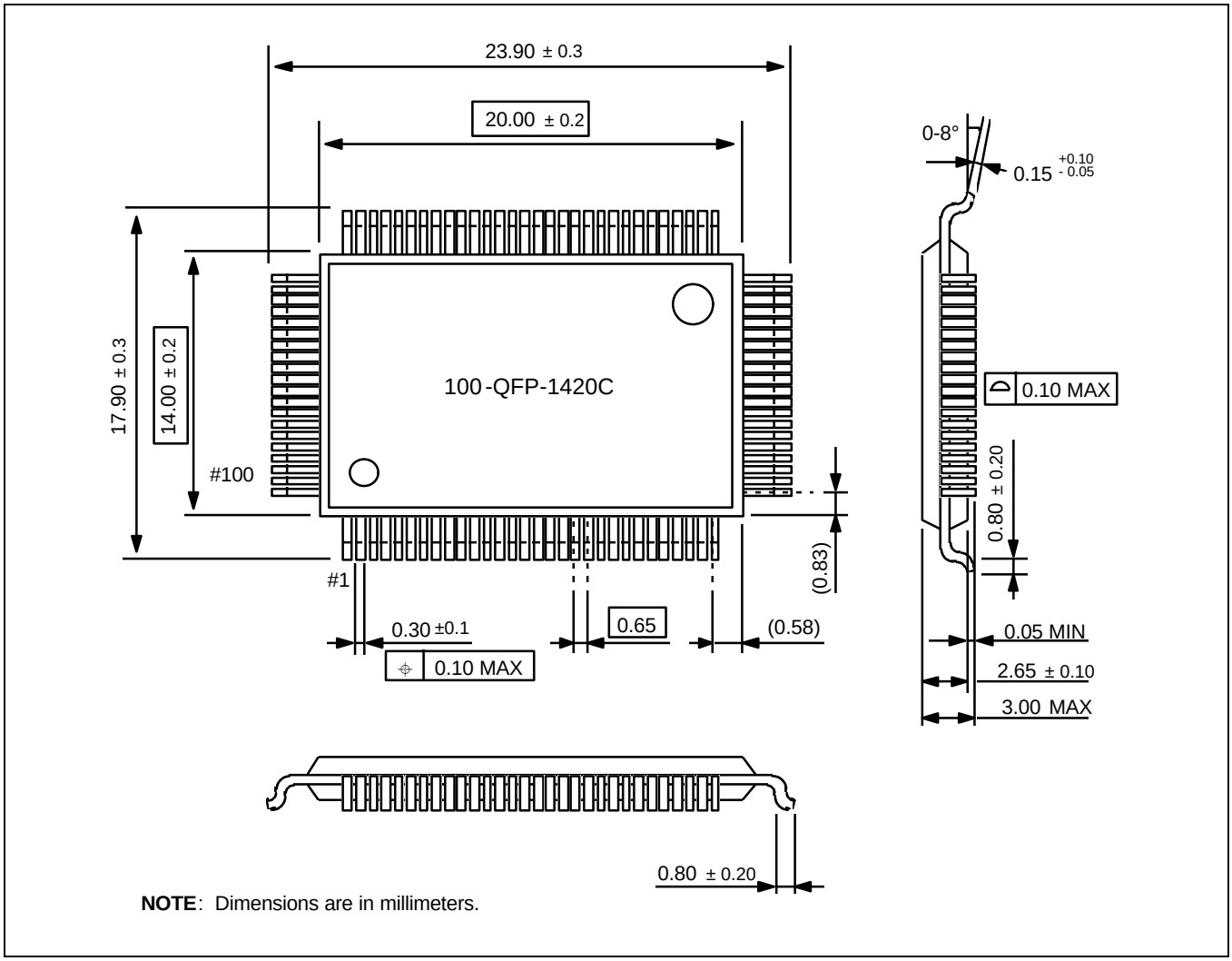


Figure 16-1. 80-Pin QFP Package Dimensions

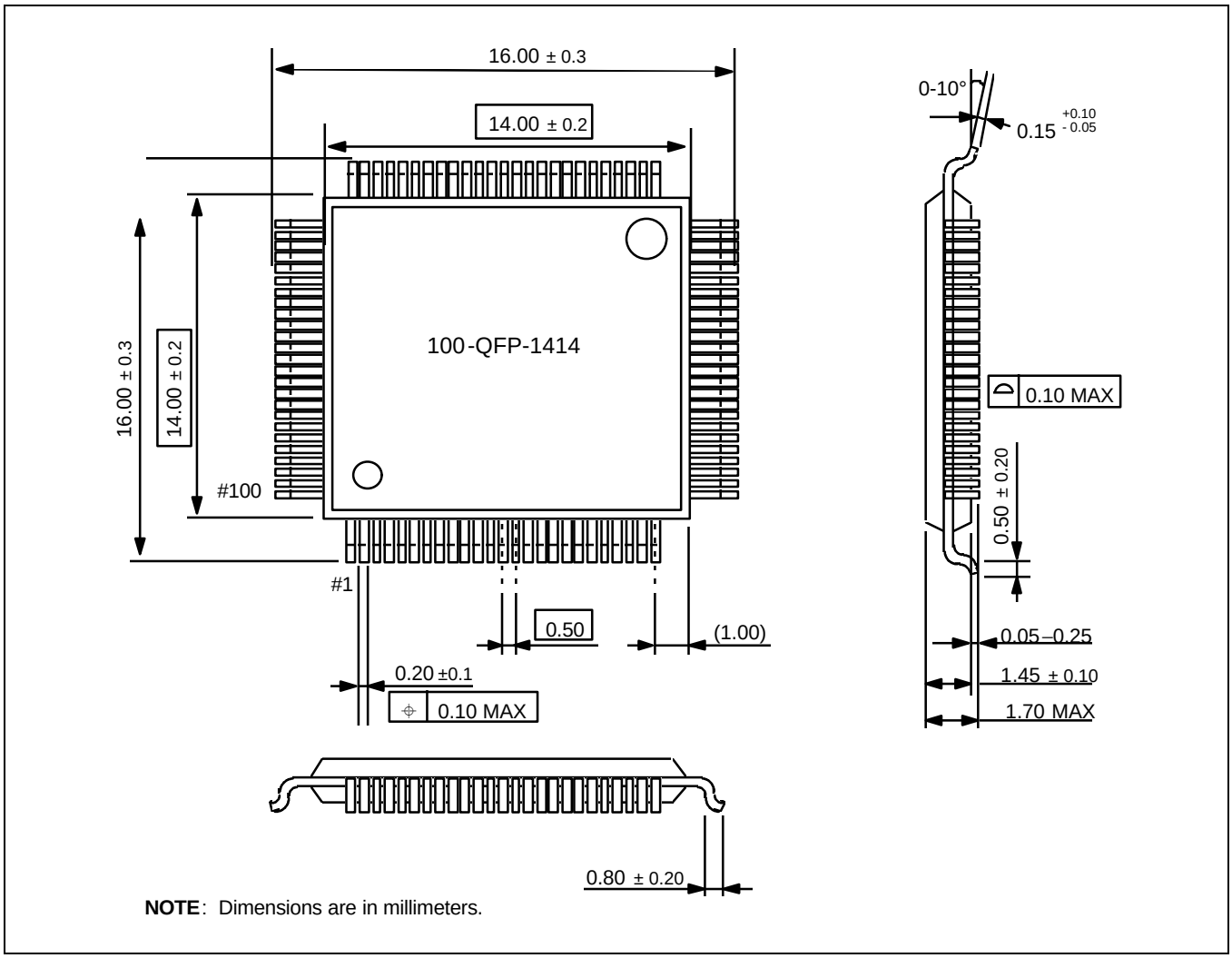


Figure 16-2. 100-Pin TQFP Package Dimensions

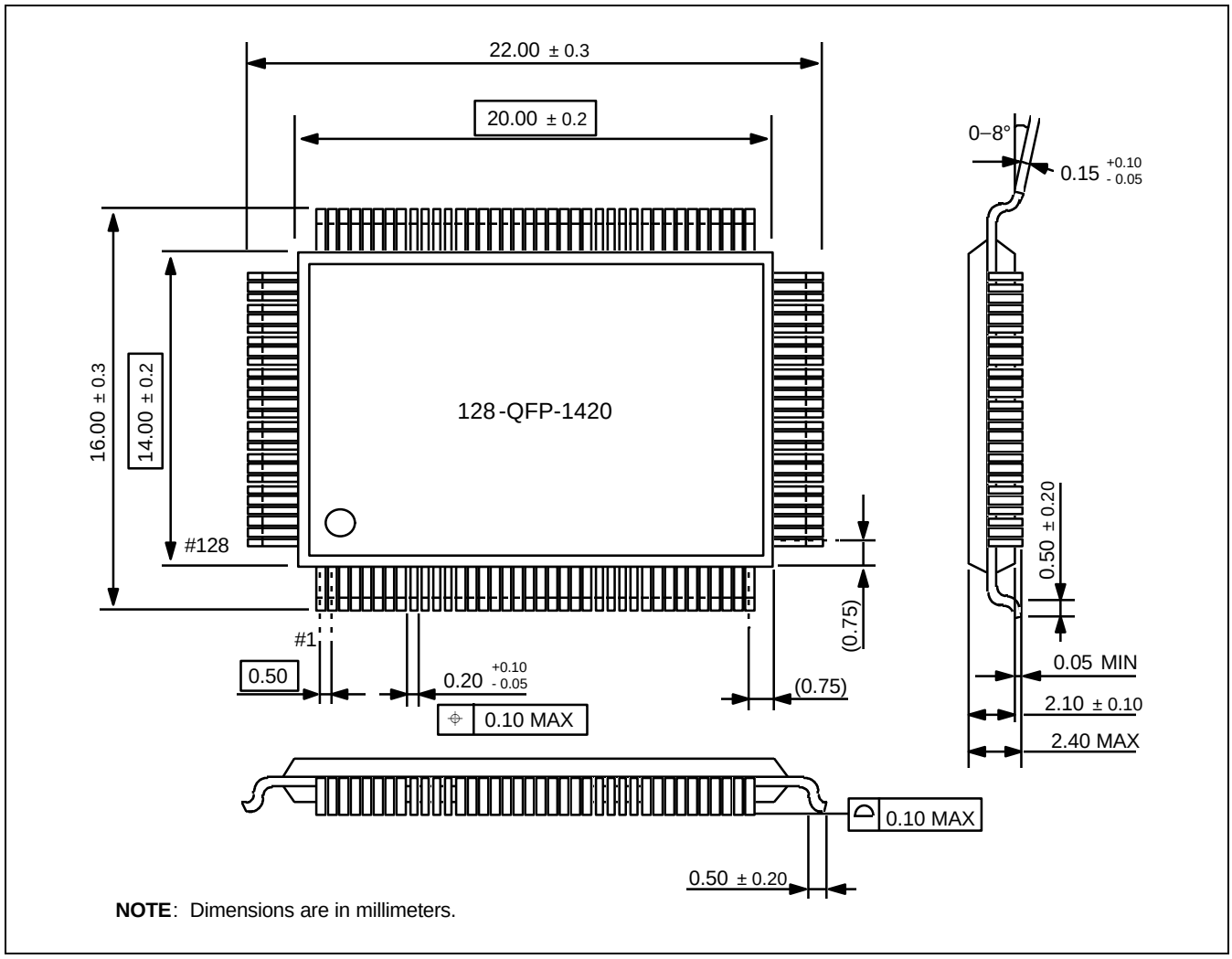


Figure 16-3. 128-QFP-1420 Package Demensions

NOTES

KS17 SERIES MASK ROM ORDER FORM

Product description:

Device Number: KS17C_____- _____(write down the ROM code number)

Product Order Form: ☐ Package ☐ Pellet ☐ Wafer Package Type: _____

Package Marking (Check One):

☐ Standard

SEC @ YWW
Device Name

☐ Custom A
(Max 10 chars)

@ YWW
Device Name

☐ Custom B
(Max 10 chars each line)

@ YWW

@ : Assembly site code, Y : Last number of assembly year, WW : Week of assembly

Delivery Dates and Quantities:

Deliverable	Required Delivery Date	Quantity	Comments
ROM code	–	Not applicable	See ROM Selection Form
Customer sample			
Risk order			See Risk Order Sheet

Please answer the following questions:

☞ **For what kind of product will you be using this order?**

☐ New product ☐ Upgrade of an existing product

☐ Replacement of an existing product ☐ Other

If you are replacing an existing product, please indicate the former product name

( )

☞ **What are the main reasons you decided to use a Samsung microcontroller in your product?**

Please check all that apply.

☐ Price ☐ Product quality ☐ Features and functions

☐ Development system ☐ Technical support ☐ Delivery on time

☐ Used same micom before ☐ Quality of documentation ☐ Samsung reputation

Mask Charge (US$ / Won): _____

Customer Information:

Company Name: _____ Telephone number _____

Signatures: _____ _____

(Person placing the order) (Technical Manager)

(For duplicate copies of this form, and for additional ordering information, please contact your local Samsung sales representative. Samsung sales offices are listed on the back cover of this book.)

KS17 SERIES
REQUEST FOR PRODUCTION AT CUSTOMER RISK

Customer Information:

Company Name: _____

Department: _____

Telephone Number: _____ Fax: _____

Date: _____

Risk Order Information:

Device Number: KS17C_____- _____ (write down the ROM code number)

Package: Number of Pins: _____ Package Type: _____

Intended Application: _____

Product Model Number: _____

Customer Risk Order Agreement:

We hereby request SEC to produce the above named product in the quantity stated below. We believe our risk order product to be in full compliance with all SEC production specifications and, to this extent, agree to assume responsibility for any and all production risks involved.

Order Quantity and Delivery Schedule:

Risk Order Quantity: _____ PCS

Delivery Schedule:

Delivery Date (s)	Quantity	Comments

Signatures: _____ _____

(Person Placing the Risk Order) (SEC Sales Representative)

(For duplicate copies of this form, and for additional ordering information, please contact your local Samsung sales representative. Samsung sales offices are listed on the back cover of this book.)

KS17C4000 MASK OPTION SELECTION FORM

Device Number: KS17C4000-_____(write down the ROM code number)

Attachment (Check one): ☐ Diskette ☐ PROM

Customer Checksum: _____

Company Name: _____

Signature (Engineer): _____

Please answer the following questions:

☞ **Application (**Product Model ID: _____)

☐ Audio	☐ Video	☐ Telecom
☐ LCD Databank	☐ Caller ID	☐ LCD Game
☐ Industrials	☐ Home Appliance	☐ Office Automation
☐ Remocon	☐ Other	

Please describe in detail its application

_____

(For duplicate copies of this form, and for additional ordering information, please contact your local Samsung sales representative. Samsung sales offices are listed on the back cover of this book.)