ARM Architecture

Sept 14 , 2005

Kyoung-Su Kim
E-mail: kimks@rayman.sejong.ac.kr
Real-Time Graphics Lab., Sejong Univ.

Outline

- Introduction
- ARM architecture
  - Architecture version & variants
  - Programmer's model
  - Instruction Set
  - ARM extension
- Coprocessor Interface
- Processor Cores
  - ARM7, ARM9, StrongARM
- AMBA
- Operating System Support
  - Memory System
  - Stack & Subroutine system
  - ARM Software development

Introduction

- Advances RISC Machines (now known as ARM) was established as a joint venture between Acorn, Apple and VLSI between Acorn, Apple and VLSI in November 1990
- ARM is the industry's leading provider of 16/32-bit embedded RISC microprocessor solutions
- The company licenses its high-performance, low-cost, power-efficient RISC processors, peripherals, and system-chip designs to leading international electronics companies
- ARM provides comprehensive support required in developing a complete system
- 32 bit RISC processor of load/store architecture

ARM architecture

- Architecture version
  - Version 1 (obsolete)
    - Basic data processing
    - Byte, word and multi-word load/store
    - Software interrupt
    - 26 bit address bus
  - Version 2 (obsolete)
    - Multiply & Multiply-accumulate
    - Coprocessor support
    - Atomic instruction for thread synchronization
    - 26 bit address bus

ARM architecture

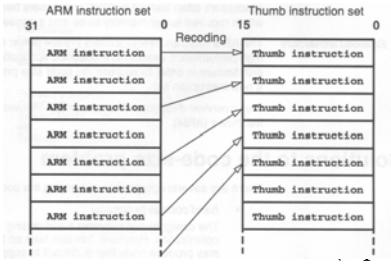
- Architecture version (cont'd)
  - Version 3
    - 32 bit address bus
    - Add CPSR, SPSR
      - Add MRS, MSR. Modify exception handler
    - Add 'Data abort mode' and 'undef mode'
  - Version 4
    - Half word transfer
    - Introduce THUMB processor state
    - Add 'Privileged mode' for operating system
    - 2 word distance of PC from current instruction
      - 'PC+8' behavior (at ARM state)
    - First fully formalized architecture

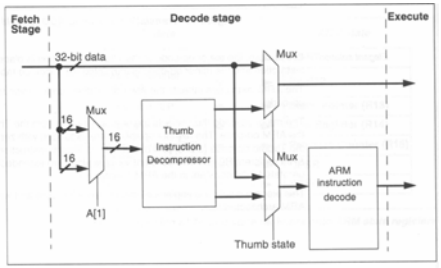
ARM architecture

- Architecture version (cont'd)
  - Version 5
    - Improve ARM/THUMB inter-working
    - Add CLZ instruction for efficient integer divide
    - Add software breakpoint
    - Add more coprocessor support
    - More tight definition of arithmetic flags

ARM architecture

- Architecture Variants
  - THUMB (symbol as a 'T')
    - THUMB instruction set: 16 bit re-encoded subset of 32 bit ARM instruction set





  - Simplified design

ARM architecture

- Architecture Variants (cont'd)
  - Long Multiply Instruction ('M' variant)
    - 32x32 = 64 bit. Provide full 64 bit result
  - Enhanced DSP instructions ('E' variant)
    - Carefully chosen addition to native ARM instruction for DSP application
    - Multiply with Q15 fixed integer. Saturation
    - 64 bit transfer
    - First introduced in v5
- Variants in Processor core
  - D: On-chip debug. Halt in response
  - I: Embedded ICE. On-chip breakpoint

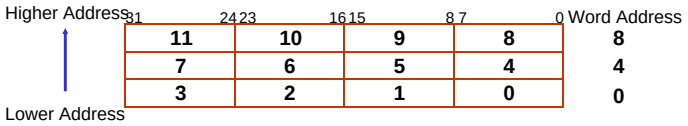
ARM architecture

- Feature of ARM programmer's model
  - 32 bit RISC processor **(32-bit data & address bus)**
  - Big and Little Endian operating modes
  - Fast interrupt response **(for real-time applications)**
  - Virtual Memory System Support
  - Excellent high-level language support
  - Simple but powerful instruction set
    - Best of RISC + Best of CISC

Programmer's Model

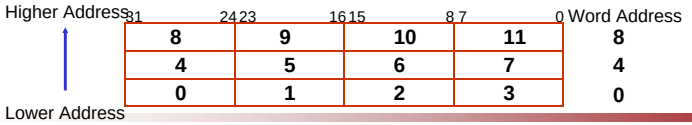
- Enianism (configured by input signal)
  - Big Endian

- Most significant byte is at lowest address
- Word is addressed by byte address of most significant byte



  - Little Endian

- Least significant byte is at lowest address
- Word is addressed by byte address of least significant byte



Programmer's Model

- Operating mode (configured by software)
  - User mode (usr)
    - the normal program execution state
  - FIQ mode (fiq)
    - designed to support a data transfer or channel process
  - IRQ mode (irq)
    - used for general purpose interrupt handling
  - Supervisor mode (svc)
    - a protected mode for the operating system
  - Abort mode (abt)
    - entered after a data or instruction prefetch abort
  - Undefined mode (und)
    - entered when an undefined instruction is executed

Programmer's Model

- Registers
  - 37 registers
    - 31 general 32 bit registers
    - 6 status registers
  - 16 general registers and one or two status registers are visible at any time
  - The visible registers depend on the processor mode
  - The other registers (the banked registers) are switched in to support IRQ, FIQ, Supervisor, Abort and Undefined mode processing
  - R0 to R15 are directly accessible
  - R0 to R14 are general purpose
  - R15 holds the Program Counter (PC)
  - CPSR - Current Program Status Register contains condition code flags and the current mode bits
  - 5 SPSRs (Saved Program Status Registers) which are loaded with CPSR when an exceptions occurs

Programmer's Model

- Registers (cont'd)

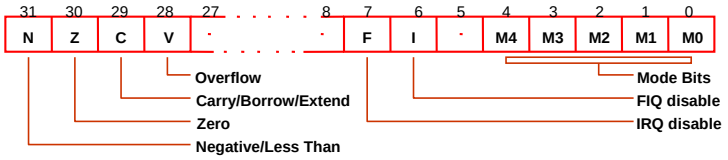
User32	Fiq32	Supervisor32	Abort32	IRQ32	Undefined32
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	R8_fiq	R8	R8	R8	R8
R9	R9_fiq	R9	R9	R9	R9
R10	R10_fiq	R10	R10	R10	R10
R11	R11_fiq	R11	R11	R11	R11
R12	R12_fiq	R12	R12	R12	R12
R13	R13_fiq	R13_svc	R13_abt	R13_irq	R13_und
R14	R14_fiq	R14_svc	R14_abt	R14_irq	R14_und
R15(PC)	R15(PC)	R15(PC)	R15(PC)	R15(PC)	R15(PC)

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

  - R13: Stack point (in common)
    - Individual stack for each processor mode
  - R14: Linked register
  - R15: Program counter

Programmer's Model

- Processor Status Registers
  - The N, Z, C and V are condition code flags
    - may be changed as a result of arithmetic and logical operations in the processor
    - may be tested by all instructions to determine if the instruction is to be executed
    - N : Negative. Z : Zero. C : Carry. V : oVerflow
  - The I and F bits are the interrupt disable bits
  - The M0, M1, M2, M3 and M4 bits are the mode bits



Programmer's Model

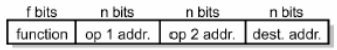
- Exceptions
  - Brake the normal execution of program
    - Handle the interrupts from peripherals
    - Guarantee the currently executed instruction in execution pipeline
  - Type of exception
    - FIQ (Fast Interrupt reQuest)
      - Externally generated by taking the nFIQ input LOW
      - Fast handling for data or channel transfer
    - IRQ(Interrupt ReQuest)
      - Normal interrupt caused by a LOW level on the nIRQ input
    - ABORT
      - Signaled by the external ABORT input
      - Indicates that the current memory access cannot be completed
    - Software interrupt
      - Generated by the software interrupt instruction (SWI)
      - Getting into Supervisor mode
      - usually to request a particular supervisor function. OS support

Programmer's Model

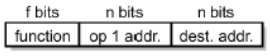
- Exceptions (cont'd)
  - Type of exception (cont'd)
    - Undefined instruction trap
      - When the ARM comes across an instruction which it cannot handle it offers it to any coprocessors which may be present
      - If a coprocessor can perform this instruction but is busy at that time, ARM will wait until the coprocessor is ready or until an interrupt occurs
      - If no coprocessor can handle the instruction then ARM will take the undefined instruction trap
  - Exception Priorities
    - (1) Reset (highest priority)
    - (2) Data abort
    - (3) FIQ
    - (4) IRQ
    - (5) Prefetch abort
    - (6) Undefined Instruction, Software interrupt (lowest priority)

ARM architecture

- Instruction Set
  - Instruction Format
    - 3 address instruction format
      - used in ARM state



    - 2 add
      - used in ARM and THUMB state



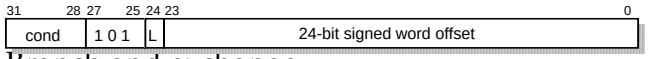
Instruction Set (cont' d)

- Conditional execution
  - All ARM instructions are conditionally executed
  - The execution may or may not take place depending on the values of the N, Z, C and V flags in the CPSR
  - All THUMB instructions are decompressed to 'Always' conditional instruction
  - Condition Field in instruction

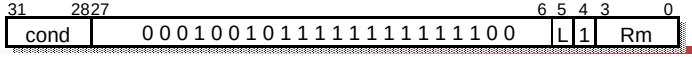
- 0000 = EQ - Z set (equal)
- 0001 = NE - Z clear (not equal)
- 0010 = CS - C set (unsigned higher or same)
- 0011 = CC - C clear (unsigned lower)
- 0100 = MI - N set (negative)
- 0101 = PL - N clear (positive or zero)
- 0110 = VS - V set (overflow)
- 0111 = VC - V clear (no overflow)
- 1000 = HI - C set and Z clear (unsigned higher)
- 1001 = LS - C clear or Z set (unsigned lower or same)
- 1010 = GE - N set and V set, or N clear and V clear (greater or equal)
- 1011 = LT - N set and V clear, or N clear and V set (less than)
- 1100 = GT - Z clear, and either N set and V set, or N clear and V clear (greater than)
- 1101 = LE - Z set, or N set and V clear, or N clear and V set (less than or equal)
- 1110 = AL - always
- 1111 = NV - never

Instruction Set (cont' d)

- Control instruction
  - Branch and branch with link
    - Jump to desired instruction
    - Save the current PC for return (with 'L' bit)

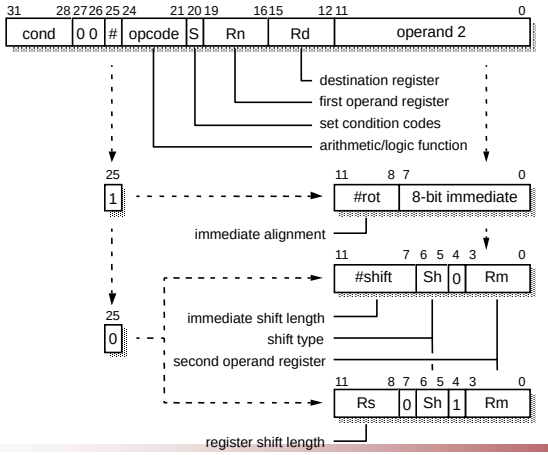


  - Branch and exchange
    - Jump to desired instruction with exchange of instruction set
      - Rm[0] == 1: Subsequent inst. are THUMB.
      - Rm[0] == 0: Subsequent inst. are ARM.



Instruction Set (cont' d)

- Data processing instruction



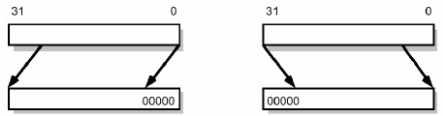
Instruction Set (cont' d)

- Data processing instruction (cont'd)

Opcode [24:21]	Mnemonic	Meaning	Effect
0000	AND	Logical bit-wise AND	Rd := Rn AND Op2
0001	EOR	Logical bit-wise exclusive OR	Rd := Rn EOR Op2
0010	SUB	Subtract	Rd := Rn - Op2
0011	RSB	Reverse subtract	Rd := Op2 - Rn
0100	ADD	Add	Rd := Rn + Op2
0101	ADC	Add with carry	Rd := Rn + Op2 + C
0110	SBC	Subtract with carry	Rd := Rn - Op2 + C - 1
0111	RSC	Reverse subtract with carry	Rd := Op2 - Rn + C - 1
1000	TST	Test	Scc on Rn AND Op2
1001	TEQ	Test equivalence	Scc on Rn EOR Op2
1010	CMP	Compare	Scc on Rn - Op2
1011	CMN	Compare negated	Scc on Rn + Op2
1100	ORR	Logical bit-wise OR	Rd := Rn OR Op2
1101	MOV	Move	Rd := Op2
1110	BIC	Bit clear	Rd := Rn AND NOT Op2
1111	MVN	Move negated	Rd := NOT Op2

Instruction Set (cont' d)

- Data processing instruction (cont'd)
  - Shift operation
    - In any data processing instructions, the second register operand can have a shift operation applied to it.
    - Logical shift

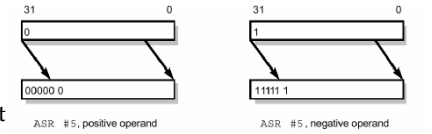


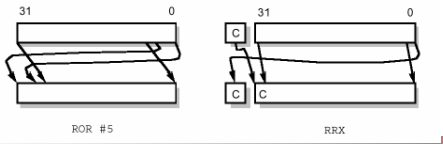
LSL #5 LSR #5

  - » Fill the vacated bits at the least significant end of the word with zeros.
  - LSR: Logical shift right by 0 to 31 places
    - » Fill the vacated bits at the most significant end of the word with zeros.

Instruction Set (cont' d)

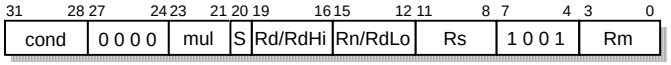
- Data processing instruction (cont'd)
  - Shift operation (cont'd)
    - Arithmetic shift
      - ASR: = LSR
      - ASL: Arithmetic shift left
        - » Sign extend the shifting bits
    - Rot





Instruction Set (cont' d)

- Multiply Instruction
  - Product two 32 bit values in registers
  - May need more cycle and power in implementation
    - Convert another instruction if possible
      - Ex) b = a * 5 : b = a + a << 2



Opcode [23:21]	Mnemonic	Meaning	Effect
000	MUL	Multiply (32-bit result)	Rd := (Rm * Rs) [31:0]
001	MLA	Multiply-accumulate (32-bit result)	Rd := (Rm * Rs + Rn) [31:0]
100	UMULL	Unsigned multiply long	RdHi:RdLo := Rm * Rs
101	UMLAL	Unsigned multiply-accumulate long	RdHi:RdLo += Rm * Rs
110	SMULL	Signed multiply long	RdHi:RdLo := Rm * Rs
111	SMLAL	Signed multiply-accumulate long	RdHi:RdLo += Rm * Rs

Instruction Set (cont' d)

- Data transfer instruction
  - Single data transfer (LDR, STR)
    - Single word(32bit), half word(16 bit) and byte(8 bit) transfer
    - Addressing
      - Register offset
        - » Address = base register ± offset register
      - Immediate offset
        - » Address = base register ± immediate constant
      - Post-indexing: modify address after use
      - Pre-indexing: modify address before use
    - Write back
      - If enable, update the base register

Instruction Set (cont' d)

- Data transfer instruction (cont'd)
  - Multiple data transfer (LDM, STM)
    - load (LDM) or store (STM) any subset of the currently visible registers
    - Use
      - Stack: maintaining full or empty stacks which can grow up or down memory
      - Context switching: Save or restore the working registers
      - Block copy: moving large blocks of data around main memory
    - Addressing
      - Pre/Post indexing
      - Auto increment or decrement
      - Write back the base register
    - Special bit
      - PSR & force user bit

Instruction Set (cont' d)

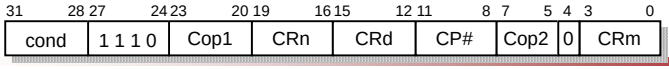
- Data transfer instruction (cont'd)
  - Single data swap (SWAP)
    - Swap a byte or word quantity between a register and external memory
    - Implemented as a memory read followed by a memory write which are "locked" together
      - Atomic instruction
      - Can't be interrupted during execution
      - External memory management unit is locked during operation by 'LOCK' signal output
    - Use
      - Synchronization in the multi-threading program (OS support)
      - Lock
      - Semaphore

Instruction Set (cont' d)

- PSR instruction (MRS, MSR)
  - The MRS and MSR instructions are formed from a subset of the Data Processing operations
  - 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

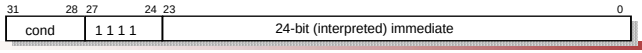
Instruction Set (cont' d)

- Coprocessor instructions
  - Coprocessor
    - General mechanism to extend the instruction set through the addition to the core
    - Example : system controller such as MMU & cache. FPU
    - Registers
      - private to coprocessor
      - ARM controls the data flow
        - » Coprocessor concerns only the data processing and memory transfer operations
  - Coprocessor data operation (CDP)
    - This class of instruction is used to tell a coprocessor to perform some internal operation
    - No result is communicated back to ARM, and it will not wait for the operation to complete



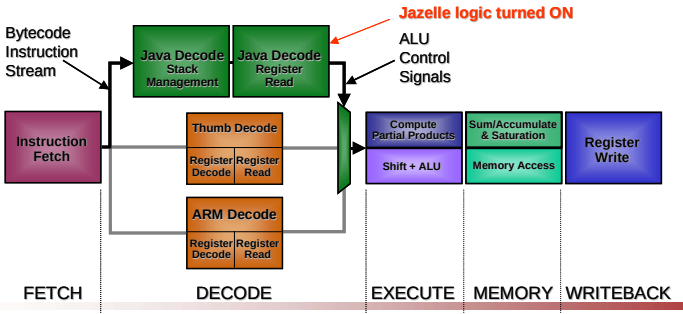
Instruction Set (cont' d)

- Coprocessor instructions (cont'd)
  - Coprocessor data transfers (LDC, STC)
    - Load (LDC) or store (STC) a subset of a coprocessors's registers directly to memory
    - ARM is responsible for supplying the memory address, and the coprocessor supplies or accepts the data and controls the number of words transferred
  - Coprocessor register transfers (MRC, MCR)
    - Communicate information directly between ARM and a coprocessor
- Software Interrupt Instruction (SWI)
  - Used to enter Supervisor mode in a controlled manner
  - The instruction causes the software interrupt trap to be taken, which effects the mode change



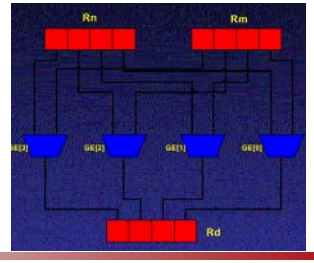
ARM architecture

- ARM extension
  - Jazelle™ : ARM's Java extension (symbol as a 'J')
    - **Instruction extension (Java state), not a coprocessor**
    - **Implemented in the ARM Pipeline as a FSM**
    - **Dynamic re-mapping of Stack to Registers**



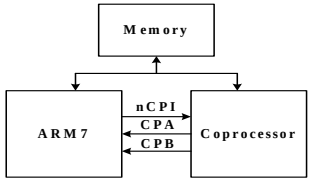
ARM architecture

- ARM extension (cont'd)
  - ARM version 6
    - Improved memory management
    - Multiprocessing
      - Add new synchronization instruction (LDREX, STREX)
    - Improved exception handling
      - New bit in PSR
    - Mixed endian support
    - Media extension
      - ARM SIMD
        - » (16bit 2 way and 8 bit 4 way)
      - FFT, MPEG4
      - Saturation, Selection…



Coprocessor Interface

- Implementation dependent
  - For ARM7 (von neuman architecture)



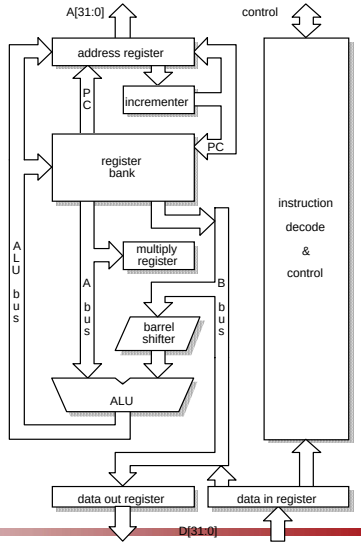
- Coprocessor present / absent (CPA)
  - nCPI LOW to execute a coprocessor instruction
  - Each coprocessor copies the instruction
  - Each coprocessor inspect the CP# field to see which coprocessor it is for
  - Every coprocessor in a system must have a unique number
  - If that number matches the contents of the CP# field the coprocessor should drive the CPA (coprocessor absent) line LOW
  - If no coprocessor has a number which matches the CP# field, CPA and CPB will remain HIGH, and ARM7 will take the undefined instruction trap

Coprocessor Interface (cont' d)

- Busy-waiting
  - If CPA goes LOW, ARM watch the CPB (coprocessor busy) line
  - ARM will busy-wait while CPB is HIGH, unless an enabled interrupt occurs
  - When CPB goes LOW, the instruction continues to completion
- Pipeline following
- Data transfer cycles
  - Coprocessor must supply or accept data at ARM7 bus rate
  - ARM7 will continue to increment the address until CPA, CPB high
- Privileged Instructions
- Idempotency
  - Any action taken by the coprocessor before it goes not-busy must be idempotent, ie must be repeatable with identical results after interrupt

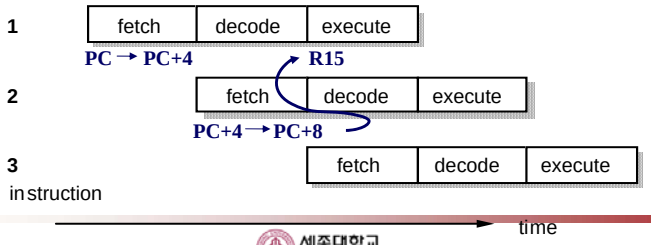
Processor Cores

- ARM7
  - Two main blocks: datapath and decoder
  - Register bank (r0 to r15)
  - Two read ports to A- bus/ B- bus
  - One write port from ALU- bus
  - Additional read/ write ports for program counter r15
  - Barrel shifter / ALU
  - Address registers/ incrementer
    - Single Memory Port
    - holds either PC address (with increment) or operand address

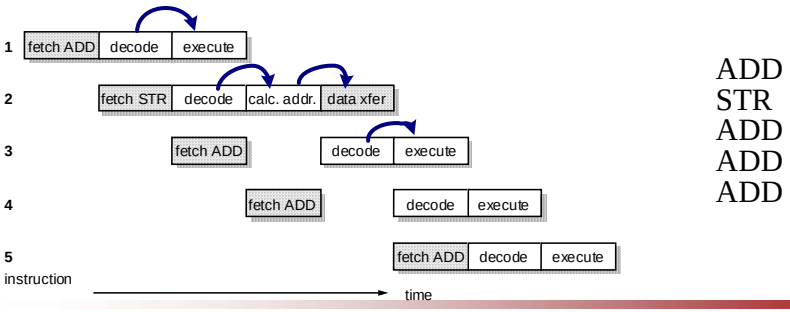


Processor Cores

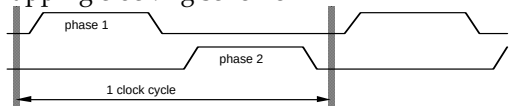
- ARM7 (cont'd)
  - Pipeline: 3 Stage pipeline
    - Fetch : fetch instruction code from memory into the instruction pipeline
    - Decode : instruction decoded to obtain control signals for the datapath ready for the next stage
    - Execute : instruction "owns" the datapath - register read; shifting; ALU results generated and write- back



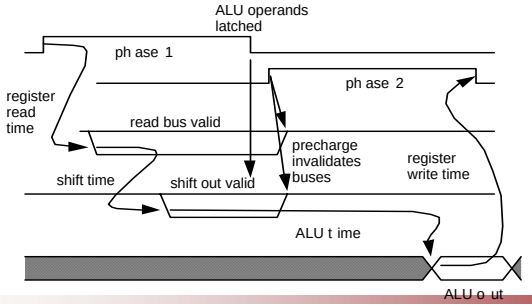
- ARM7(cont'd)
  - Multi-cycle operation
    - Single cycle throughput for almost simple data processing instruction
    - Multi-cycle for mul, load/store



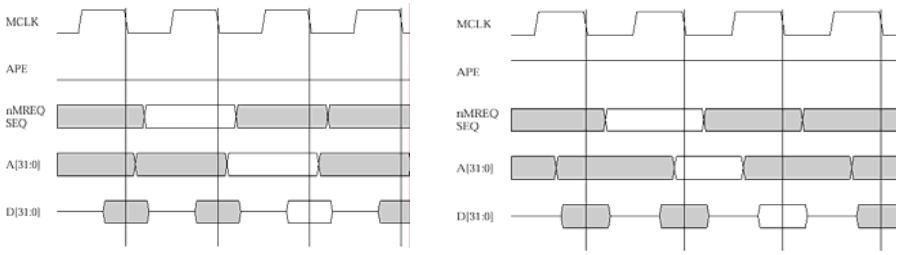
- ARM7(cont'd)
  - 2 Phase Non-overlapping clocking scheme



  - Datapath timing



- ARM7(cont'd)
  - Memory Interface

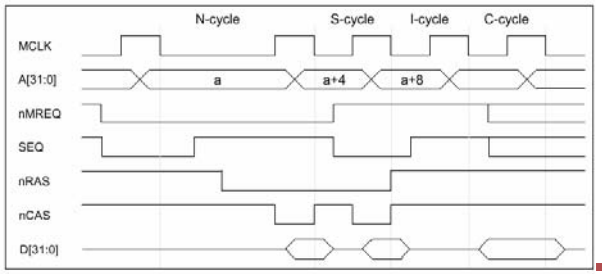


De-pipelined addressing Pipelined addressing

  - mREQ must be valid before actual reference cycle

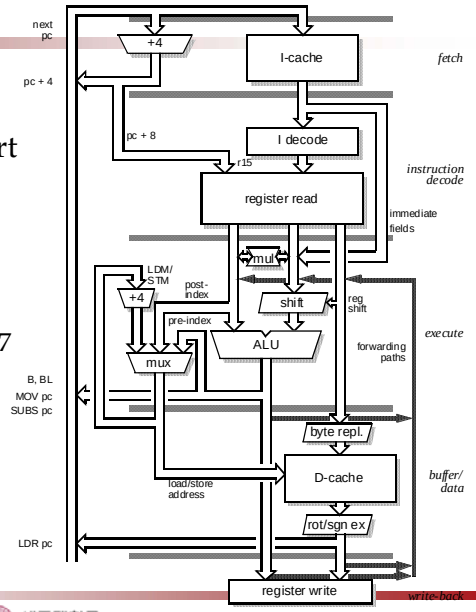
- ARM7(cont'd)
  - Cycle type

nMREQ	SEQ	Cycle type
0	0	Non-sequential (N-cycle)
0	1	Sequential (S-cycle)
1	0	Internal (I-cycle)
1	1	Coprocessor register transfer (C-cycle)



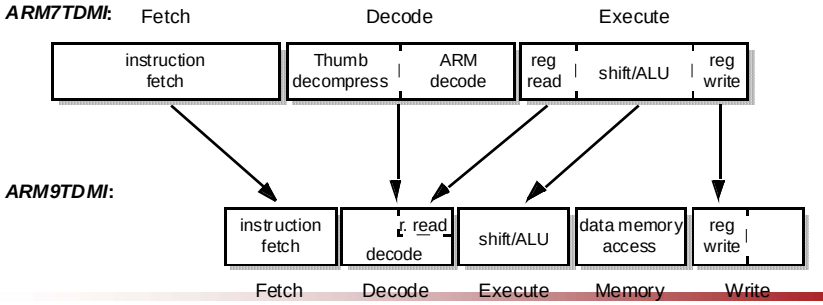
Processor Cores

- ARM9
  - Separate memory port for high CPI
    - Instruction
    - Data
  - Datapath
    - Almost same as ARM7
    - Compatible to ARM7



Processor Cores

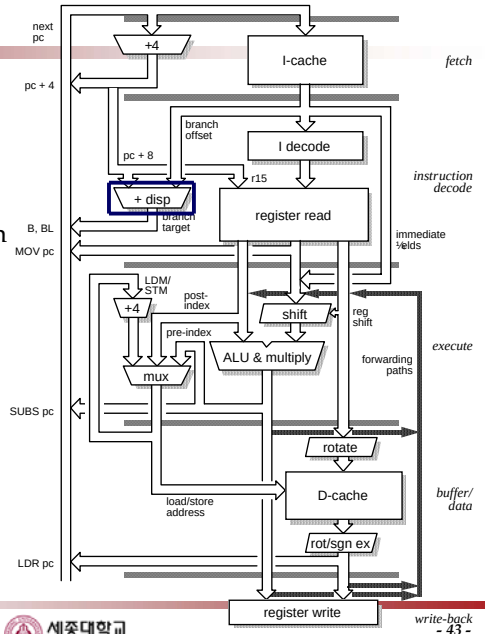
- ARM9(cont'd)
  - 5 Stage Pipeline
  - Multi-cycle operation: MUL, multiple load/store
  - Data forwarding



Processor Cores

- StrongARM
  - Harvard architecture
    - Separate instruction and data port
  - 5 Stage pipeline same as ARM9
  - First developed by DEC, now Intel

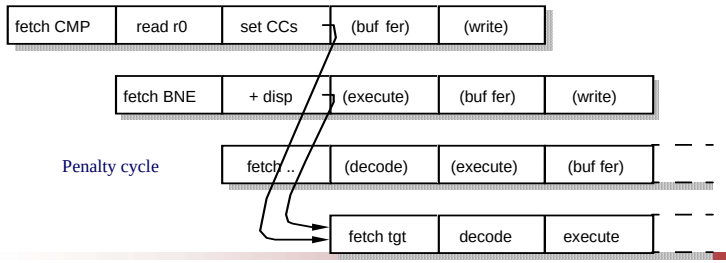
*SA1110: v4
*XScale: v5TE



Processor Cores

- StrongARM(cont'd)
  - Branch target adder in decode stage
    - Reduce the taken branch penalty to 1 cycle
      - Applied B, BL and return from function call

```
CMP r0, #0
BNE label
…
```



Processor Cores

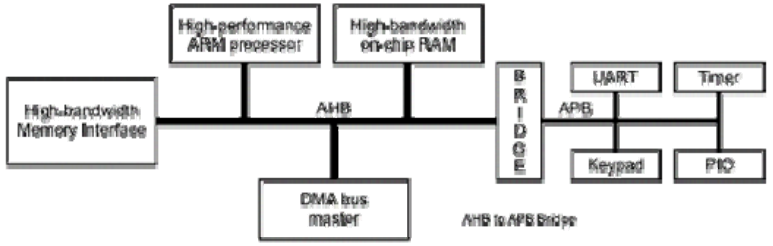
- StrongARM(cont'd)
  - Multiply implementation

	Memory port	Multiplier	Branch adder
ARM7	1	8 bit	x
ARM9	2	8 bit	x
StrongARM	2	12 bit	0

  - Reduce the issue latency of MUL to 1 ~3 cycle
    - Compared to ARM7, ARM9 (1~4 cycle)

AMBA

- Advanced Microcontroller Bus Architecture
  - Standard of on-chip communication between different macrocells for high performance embedded system design
  - Hierarchical Bus architecture



AMBA

- AMBA buses
  - AHB(Advanced High Performance Bus)
    - Connect between high-performance system modules
  - ASB(Advanced System Bus)
    - Subset of AHB
  - APB(Advanced Peripheral Bus)
    - Simple interface for low-performance peripherals

AMBA

- AMBA buses(cont'd)

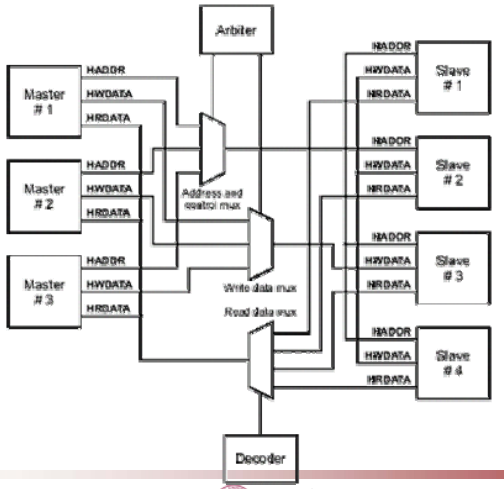
AHB	ASB	APB
- burst transfers - split transactions - single-cycle bus master handover - single-clock edge operation - wider data bus configurations (64/128 bits) - multiple bus masters (up to 16) - pipelined operation	- burst transfers - pipelined operation - multiple bus masters	- low power - latched address and control - simple interface - suitable for many peripherals

AMBA

- AMBA AHB component
  - Master
    - Initiate read and write operations by providing an address and control information. Only one bus master is allowed to actively use the bus at any one time.
  - Slave
    - Responds to a read or write operation within a given address-space range. The bus slave signals back to the active master the success, failure or waiting of the data transfer.
  - Arbiter
    - Ensures that only one bus master at a time is allowed to initiate data transfers. Can use the priority
  - Decoder
    - Decode the address of each transfer and provide a select signal for the slave that is involved in the transfer.

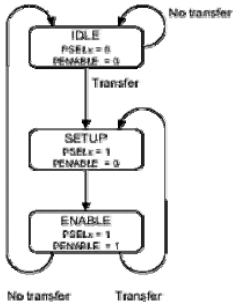
AMBA

- AMBA AHB component (cont'd)



AMBA

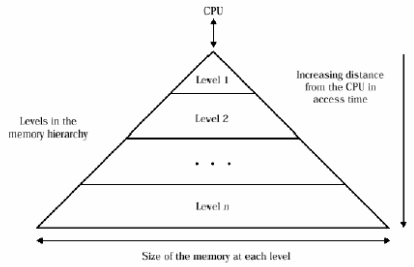
- AMBA APB
  - APB bridge: only master in APB. Act as slave in AHB
  - APB slave: peripherals
  - Simple protocol

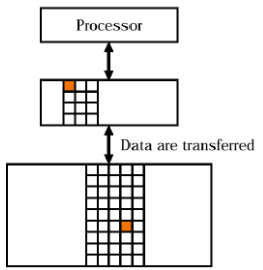


- Processor core
  - Master in AHB
  - Connect through the memory interface of core

Operating System Support

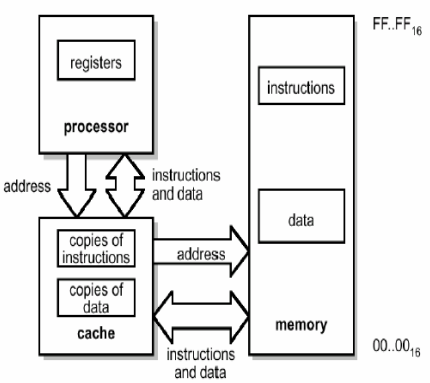
- Memory System
  - Memory hierarchy
    - Cache system
      - Temporal locality
      - Spatial locality



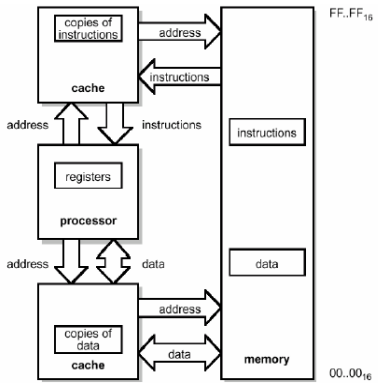


Cache system (cont' d)

- Single cache shared between instruction and data



- **Separate data and instruction cache**



Cache system (cont' d)

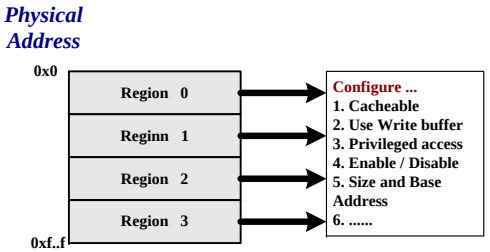
- Write strategy
  - Write- through
    - All write are passed to main memory immediately
    - If there is a hit, the cache is updated to hold new value
    - Processor slow down to main memory speed during write
  - Write- through with buffered write
    - Use a buffer to hold data to write back to main memory
    - Processor only slowed down to write buffer speed (which is fast)
    - Write buffer transfers data to main memory (slowly), processor continues its tasks
  - Copy- back
    - Write operation updates the cache, but not main memory
    - Cache remember that it is different from main memory via a dirty bit
    - It is copied back to main memory only when the cache line is used by new data

Operating System Support

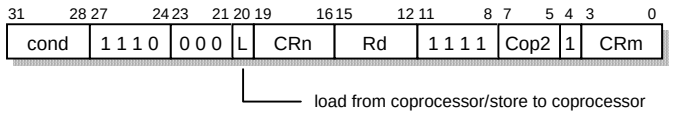
- Memory System (cont'd)
  - CP15 system control coprocessor
    - On-chip coprocessor which controls the on-chip cache, memory management and other system configuration signals
    - Mapped as a coprocessor of number 15
    - Protection unit
      - Embedded system with fixed and controlled application
        - » Not need full virtual memory system
        - » Stand-alone mp3 player…
    - Memory Management Unit
      - General purpose application where the range and number of programs is unknown at design time
        - » Need virtual memory system support
        - » PDA…

Operating System Support

- Protection Unit

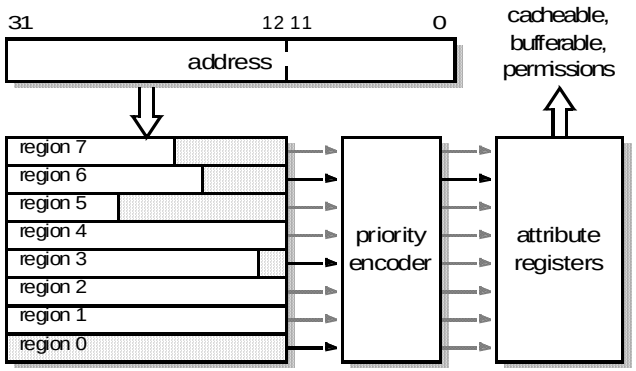


Register	Purpose
0	ID Register
1	Configuration
2	Cache Control
3	Write Buffer Control
5	Access Permissions
6	Region Base and Size
7	Cache Operations
9	Cache Lock Down
15	Test
4, 8, 10-14	UNUSED



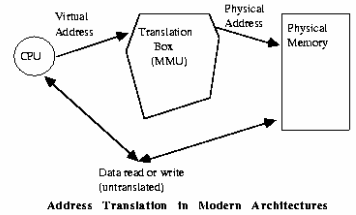
Operating System Support

- Protection Unit (cont'd)
  - ARM protection unit



Operating System Support

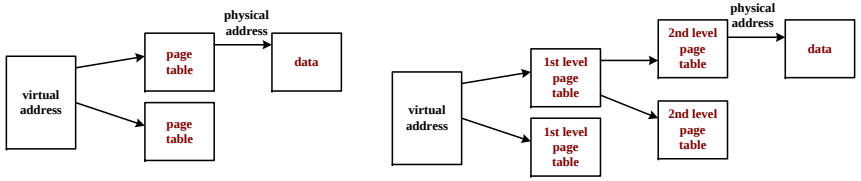
- Memory Management Unit
  - Virtual memory system



Address Translation in Modern Architectures

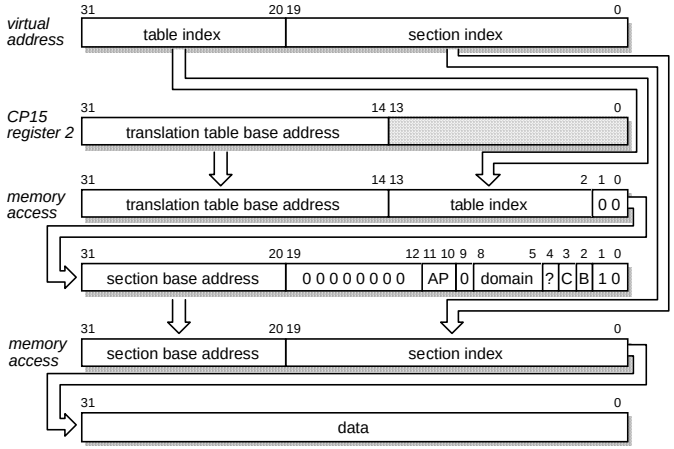
Memory Management Unit (cont' d)

- ARM MMU
  - Translates virtual address to physical address
  - Controls memory access permission
  - Use 2 level page table with TLB
    - Page: fixed size of chunk of memory
    - TLB: cache of virtual to physical address mapping



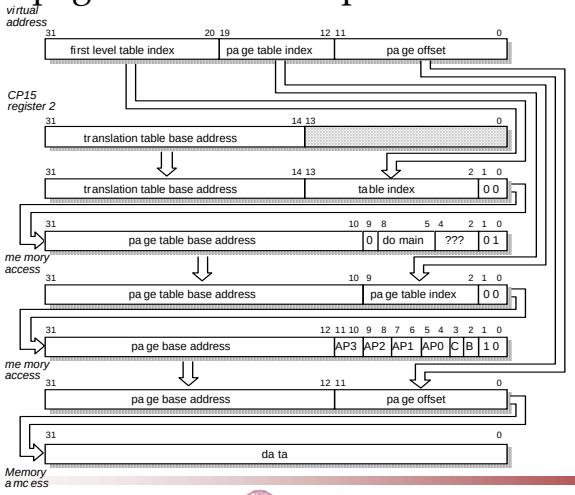
Memory Management Unit (cont' d)

- Selection translation sequence



Memory Management Unit (cont' d)

- Small page translation sequence



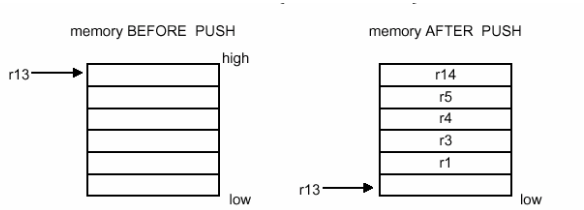
Memory Management Unit (cont' d)

- CP15 control registers

Register	Purpose
0	ID Register
1	Control
2	Translation Table Base
3	Domain Access Control
5	Fault Status
6	Fault Address
7	Cache Operations
8	TLB Operations
9	Read Buffer Operations
10	TLB lockdown
13	Process ID Mapping
14	Debug Support
15	Test & Clock Control
4, 11-12	UNUSED

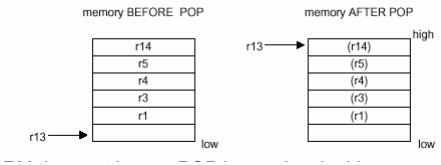
Operating System Support

- Stack and Subroutine System
  - Idea of stack
    - The multiple load/ store instructions can be used to implement last-in- first- out storage called a STACK .
    - A stack is a portion of main memory used to store data temporarily
    - A PUSH operation which stores a number of registers onto the stack memory.

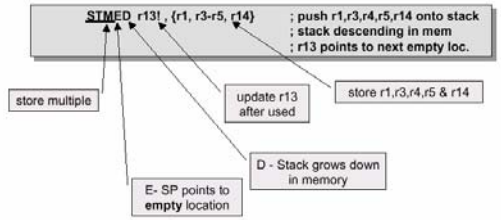


Stack & Subroutine System (cont' d)

- Stack(cont'd)
  - Pop operation

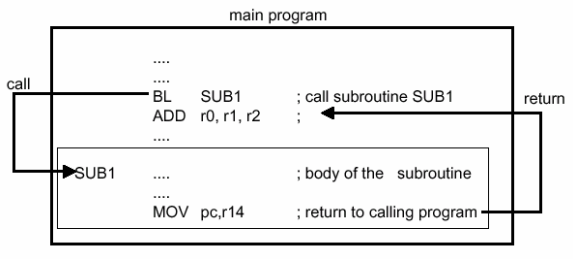


  - Implemented by LDM/STM instruction



Stack & Subroutine System (cont' d)

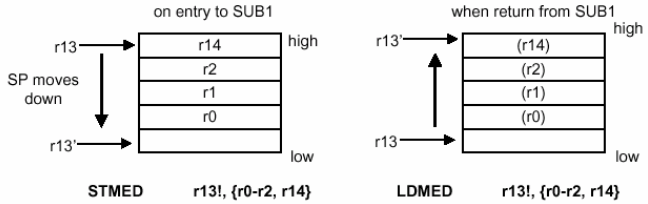
- Subroutine
  - Subroutines allow you to modularize your code so that they are more reusable.



Stack & Subroutine System (cont' d)

- Subroutine with saving the context

```
        BL      SUB1
        .....
SUB1    STMED   r13!, {r0-r2, r14}      ; push work & link registers
        ....
        BL      SUB2                    ; jump to a nested subroutine
        ...
        LDMED   r13!, {r0-r2, r14}      ; pop work & link registers
        MOV     pc, r14                 ; return to calling program
```



Operating System Support

- ARM Software Development
  - ARM software development tookit
    - armcc, armasm, armlink
  - ARMulator
    - Cycle accurate simulator
    - MMU, coprocessor
    - Profiler
  - Boot-up code
    - On reset , processor starts at address 0x0
  - ARM procedure call standard
    - Can inter-work assembly routine with C/C++ program