ARM Processors

-Swarna Prabha Jena
Department of ECE

ARM Powered Products

The History of ARM

- Developed at Acorn Computers Limited, of Cambridge, England, between 1983 and 1985
- Problems with CISC:
  - Slower then memory parts
  - Clock cycles per instruction

The History of ARM (2)

- Solution - the Berkeley RISC I:
  - Competitive
  - Easy to develop (less than a year)
  - Cheap
  - Pointing the way to the future

Why learn ARM?

- Dominant architecture for embedded systems
- 32 bits=> powerful & fast
- Efficient: very low power/MI PS
- Regular instruction set with many advanced features.

Beyond MUO - A first look at ARM

- **Complete instruction set.**
- **Larger address**
- **Subroutine call mechanism**
- **Additional internal registers**
- **Interrupts, direct memory access (DMA), and cache memory.**

- ***Interrupts:*** allow external devices (e.g. mouse, keyboard) to interrupt the current program execution
- ***DMA:*** allows external high• throughput devices (e.g. display card) to access memory directly rather than through processor
- **Cache:** a small amount of fast memory on the processor

The ARM Instruction Set

- Load-Store architecture
- Fixed-length (32-bit) instructions
- 3-operand instruction format (2 source operand regs, 1 result operand reg): ALU operations very powerful (can include shifts)
- Conditional execution of ALL instructions (v. clever idea!)
- Load-Store multiple registers in one instruction
- A single-cycle n-bit shift with ALU operation
- "Combines the best of RISC with the best of CISC"

Operating Modes

User mode

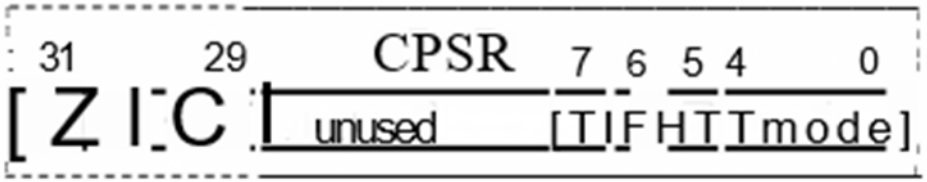
- Normal program execution mode
- System resources unavailable
- Mode can be changed by supervisor only

Supervisor modes

- Entered upon exception
- Full access to system resources
- Mode changed freely

ARM Programmer's Model

- 16 X 32 bit registers
- R15 is equal to the PC
  - Its value is the current PC value
  - Writing to it causes a branch!
- RO-R14 are general purpose
  - R13, R14 have additional functions, described later
- Current Processor Status Register (CPSR)
  - Holds condition codes i.e status bits



ARM Visible Registers

ARM Programmer's Model

- CPSR is a special register, it cannot be read or written like other registers
  - The result of any data processing instruction can modify **status bits (flags)**
  - These flags are read to determine branch conditions etc

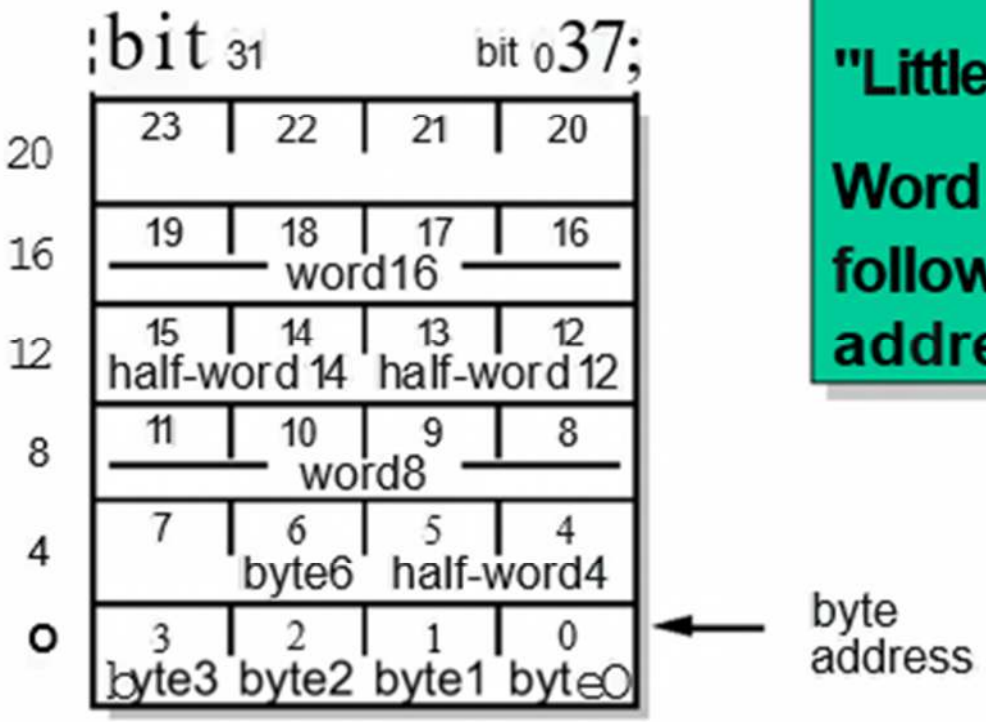
  Main status bits **(condition codes):**
  - N (result was negative)

    Z (result was zero)
  - C (result involved a carry-out)
  - V (result overflowed as signed number)
- Other fields described later

ARM CPSR format

ARM's **memory organization**

- Byte addressed memory
- Maximum 2^{32} bytes of memory
- A word = 32-bits, half-word = 16 bits
- Words aligned on 4-byte boundaries

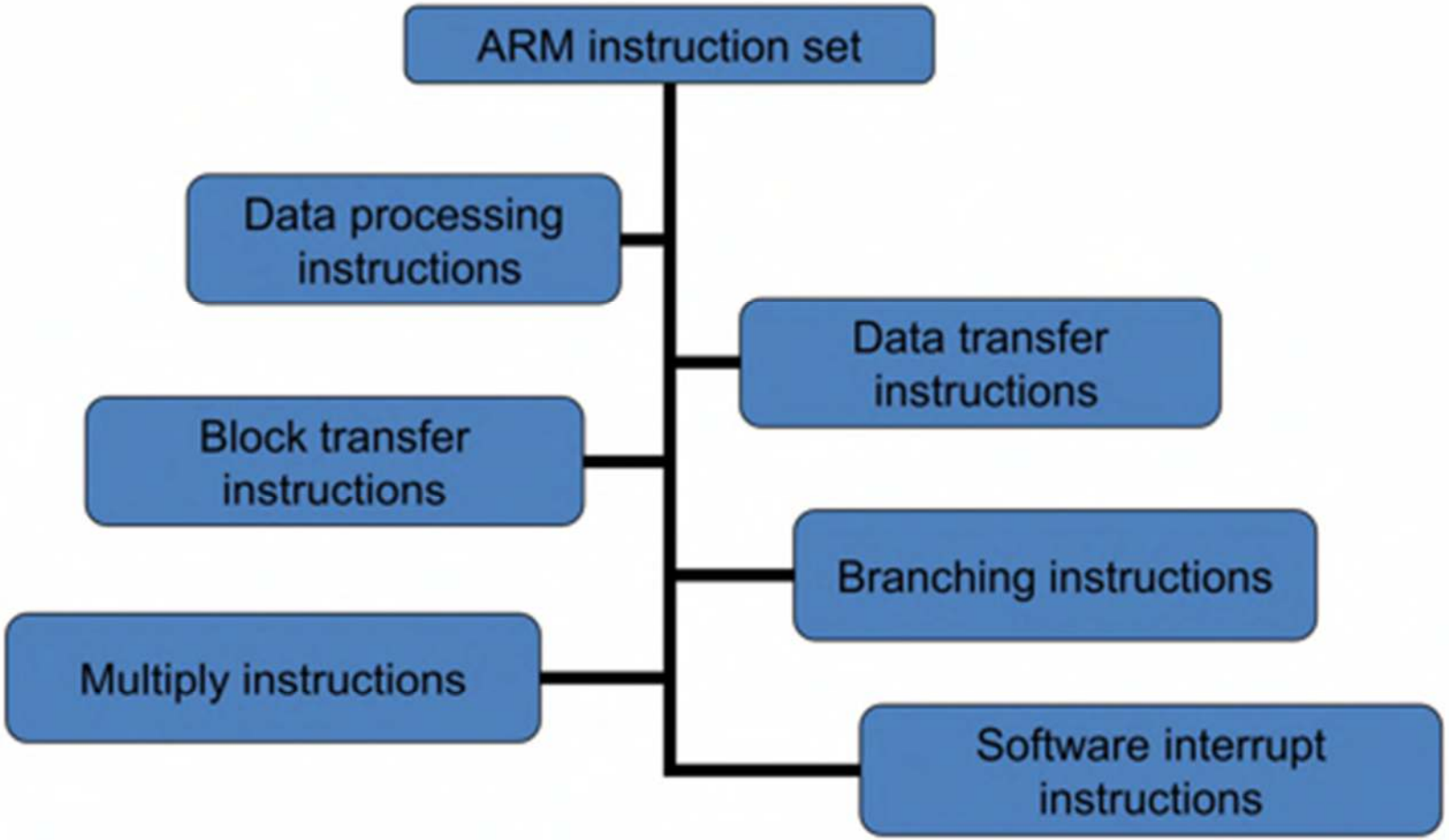


NB - Lowest byte address = LSB of word

"Little-end ian"

Word addresses follow LSB byte address

ARM Instruction Set (3)



Data Processing Instructions

- Arithmetic and logical operations
- 3-address format:
  - Two 32-bit operands
    (opl is register, op2 is register or immediate)
  - 32-bit result placed in a register
- Barrel shifter for op2 allows full 32-bit shift within instruction cycle

Data Processing Instructions (2)

- Arithmetic operations:
  - ADD, ADDC, SUB, SUBC, RSB, RSC
- Bit-wise logical operations:
  - AND, EOR, ORR, BIC
- Register movement operations:
  - MOV, MVN
- Comparison operations:
  - TST, TEQ, CMP, CMN

Data Processing Instructions (3)

Conditional codes

+

Data processing instructions

+

Barrel shifter

-

Powerful tools for efficient coded programs

Data Processing Instructions (4)

Example

if (z==1)

R1=R2+(R34)

compiles to

EQADDS R1,R2,R3, LSL #2

(SINGLE INSTRUCTION !)

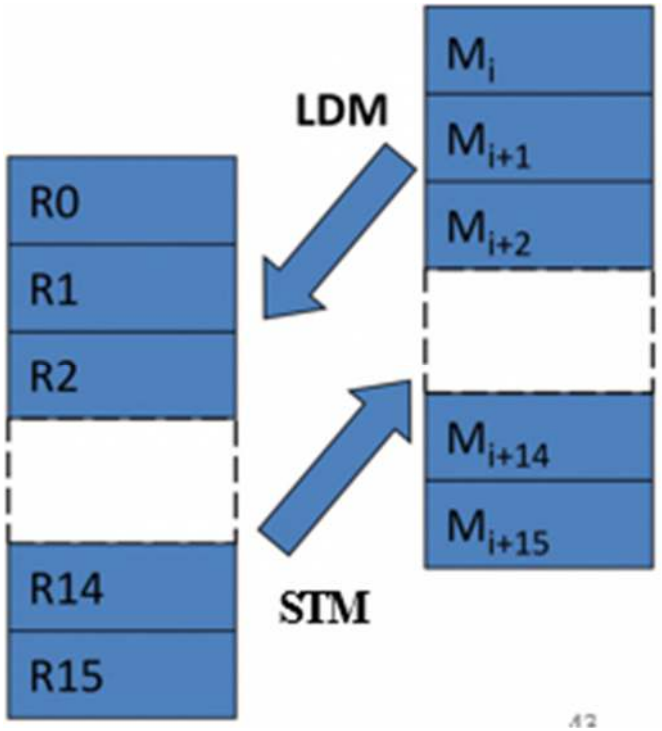
Data Transfer Instructions

- Load/store instructions
- Used to move signed and unsigned
  Word, Half Word and Byte to and from registers
- Can be used to load PC
  (if target address is beyond branch instruction range)

LDR	**Load Word**	**STR**	**Store Word**
LDRH	**Load Half Word**	**STRH**	**Store Half Word**
LDRSH	**Load Signed Half Word**	**STRSH**	**Store Signed Half Word**
LDRB	**Load Byte**	**STRB**	**Store Byte**
LDRSB	**Load Signed Byte**	**STRSB**	**Store Signed Byte**

Block Transfer Instructions

- Load/Store Multiple instructions ***(LDM/STM)***
- Whole register bank or a subset copied to memory or restored with single instruction

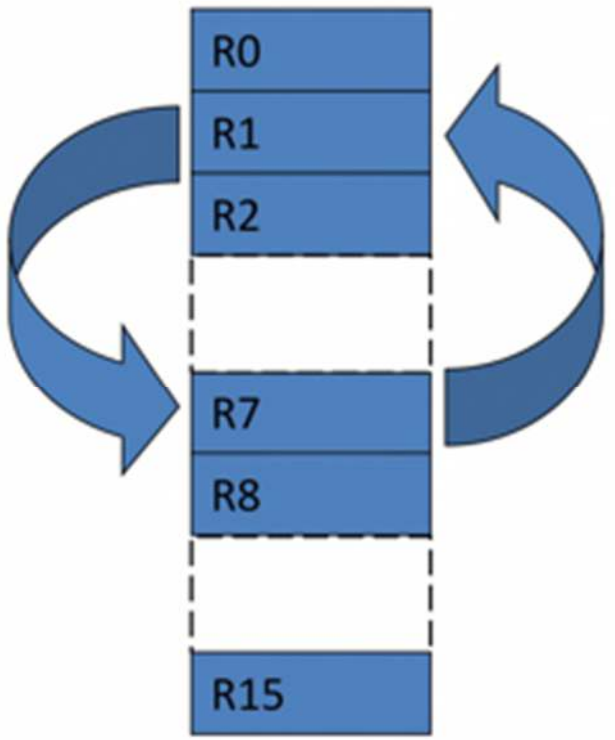


Swap Instruction

- Exchanges a word between registers
  - Two cycles

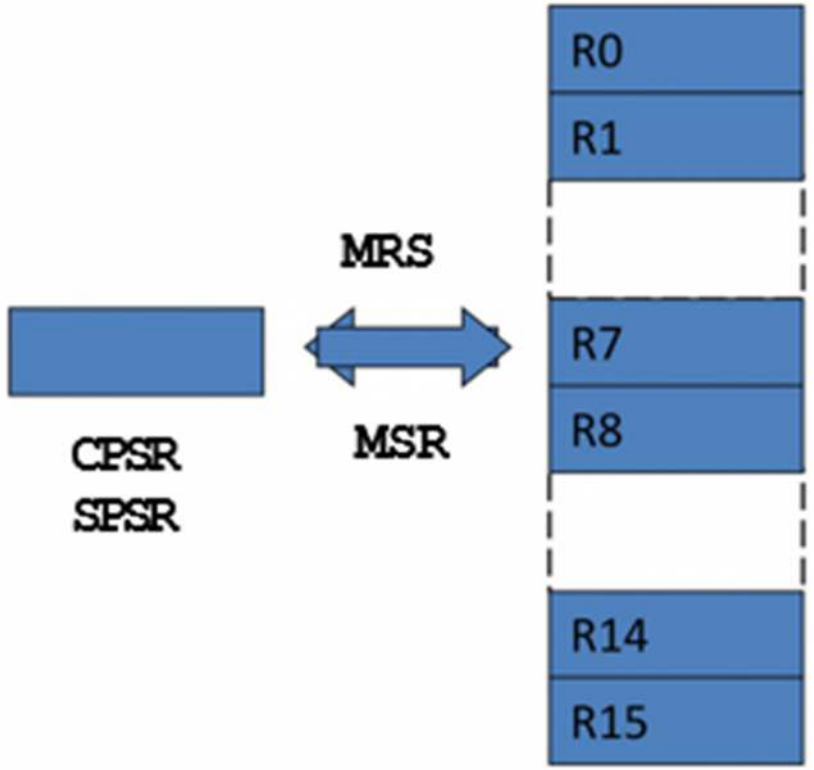
    but

  single atomic action
  - Support for RT semaphores



Modifying the Status Registers

- Only indirectly
- *MSR* moves contents fram CPSR/SPSR to selected GPR
- *MRS* moves contents from selected GPR to CPSR/SPSR
- Only in privileged modes



Multiply Instructions

- Integer multiplication (32-bit result)
- Long integer multiplication (64-bit result)
- Built in Multiply Accumulate Unit (MAC)
- Multiply and accumulate instructions add product to running total

Multiply Instructions

- Instructions:

MUL	Multiply	32-bit result
MULA	Multiply accumulate	32-bit result
UMULL	Unsigned multiply	64-bit result
UMLAL	Unsigned multiply accumulate	64-bit result
SMULL	Signed multiply	64-bit result
SMLAL	Signed multiply accumulate	64-bit result

Branching Instructions

- *Branch* (B):

  jumps forwards/backwards

  up to 32 MB

- *Branch link* (BL):

same+ saves (PC+4) in LR

- Suitable for function call/return

- Condition codes for conditional branches

Branching Instructions (2)

- *Branch exchange (BX)* and
  Branch link exchange (BLX):
  same as *B/BL* + exchange instruction set (ARM ++THUMB)
- Only way to swap sets

Thumb Instruction Set

- Compressed form of ARM
  - Instructions stored as 16-bit,
  - Decompressed into ARM instructions and
  - Executed
- Lower performance (ARM 40% faster)
- Higher density (THUMB saves 30% space)
- Optimal -

  "interworking" (combining two sets) - compiler supported

THUMB Instruction Set (2)

- More traditional:
  - No condition codes
  - Two-address data processing instructions
- Access to RO - R8 restricted to
  - *MOV, ADD, CMP*
- PUSH/POP for stack manipulation
  - Descending stack (SP hardwired to R13)

THUMB Instruction Set (3)

- No *MSR* and *MRS*, must change to ARM to modify CPSR (change using *BX* or *BLX)*
- ARM entered automatically after RESET or entering exception mode
- Maximum 255 SWI calls

ARM Assembly Quick Recap

MOV ra, rb	ra := rb
MOV ra, #n	ra := n
ADD ra, rb, re	ra := rb + re
ADD ra, rb, #n	ra := rb + n
CMP ra, rb	set status bits on ra-rb
CMP ra, #n	set status bits on ra-n
B label	branch to label
BEQ label	branch to label if zero
BNE label	branch if not zero
BMI label	branch if negative
BPL label	branch if zero or plus
LDR ra, label	ra := mem[label]
STR ra, label	mem[label] := ra
ADR ra, label	ra :=address of label
LDR ra, [rb]	ra := mem[rb]
STR ra, [rb]	mem[rb] := ra

- n decimal in range -128 to 127 **(other values possible, see later)**
- ♦SUB => - instead of+
- CMP is like SUB but has no destination register ans sets status bits
- **BL label** is branch & link
- Branch conditions apply to the result of the last instruction to set status bits (ADDS/SUBS/MOVS/CMP etc).
- ♦LDRB/STRB => byte transfer
- ♦Other address modes:

[rb,#n] => **mem[rb+n]**
[rb,#n]! => **mem[rb+n], rb** := **rb+n**
[rb],#n => **mem[rb], rb:=rb+n**
[rb+ri] => **mem[rb+ri]**