UNIT III 32-BIT CONTROLLER

Fundamentals – Instruction set - Thumb Instruction set - Writing and Optimizing - assembly codes - Efficient C programming - Optimized Primitives - Digital Signal Processing - Exception and Interrupt Handling - Firmware.

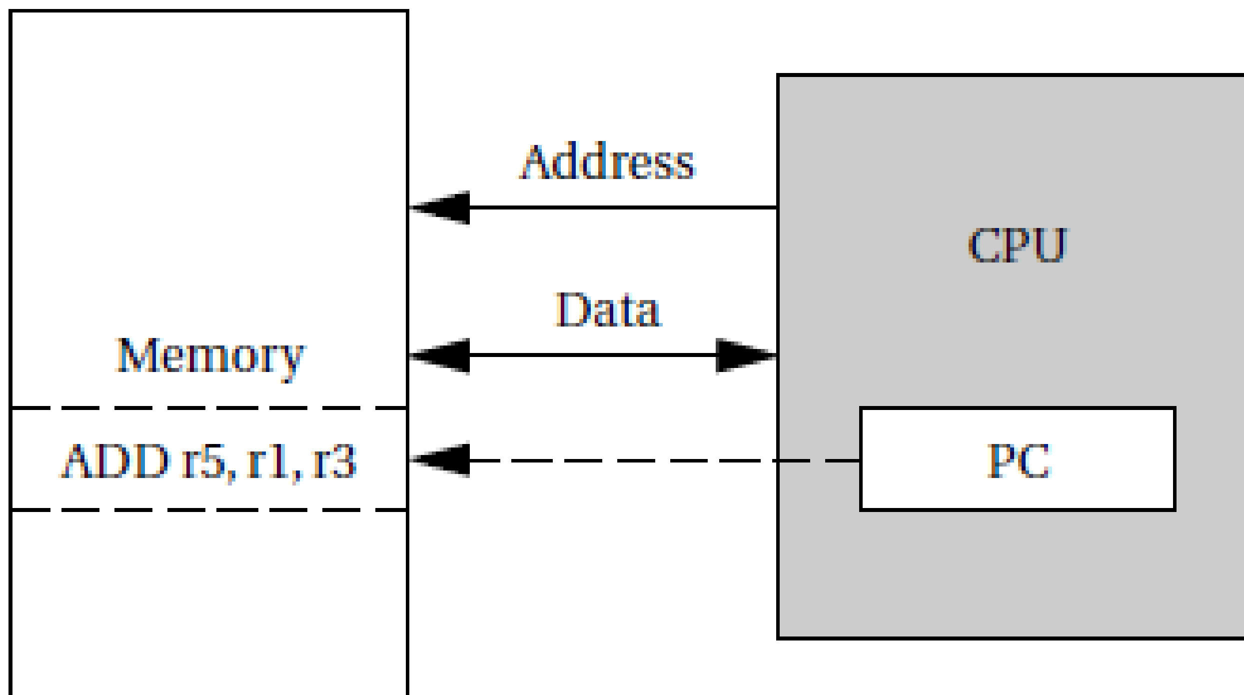
Textbooks

1. Wayne Wolf, “Computers as Components - Principles of Embedded Computing System Design”, Morgan Kaufmann Publisher (An imprint from Elsevier), Second Edition ,2008.

2. Andrew N Sloss, Dominic Symes, Chris Wright, “ARM System Developer’s Guide- Designing and Optimizing System Software”, Elsevier/Morgan Kaufmann Publisher, 2008.

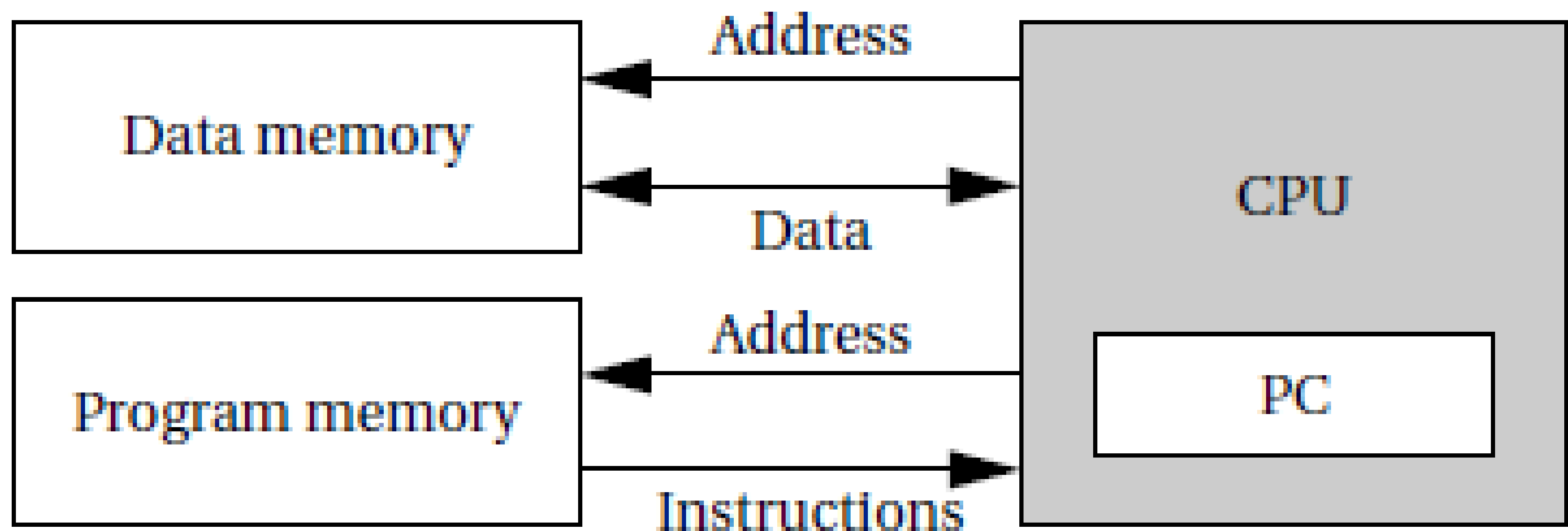
Types of Memory Architecture- Von Neumann Architecture

- Program Counter (PC) holds the memory address of the next instruction, which has to be executed by CPU.
- The CPU fetches the instruction from memory, decodes the instruction, and executes it.
- Eg: ARM 7



Types of Memory Architecture- Harvard Architecture

- Harvard machine has separate memories for data and program (Eg: ARM 9)
- The program counter points to program memory, not data memory
- Provides higher performance for digital signal processing (Eg: Streaming data)



Computer Architecture

- Relates to their instructions and how they are executed
- Complex instruction set computers (CISC): large no of instructions, addressing modes and instruction formats of varying length.
- Reduced instruction set computers (RISC): Provide fewer and simpler instructions. Instructions could be efficiently executed in pipelined processors
- Computers can be classified based on the characteristics of their instruction sets such as,
  - Fixed versus variable length
  - Number of operands
  - Types of operations supported
  - Addressing modes

Dr. V. Sathiesh Kumar Department of Electronics Engineering, MIT, India

History of ARM Processor

- ARM Processor- 32 bit processor
- RISC (Reduced Instruction Set Computers) concept introduced in 1980 at Stanford and Berkley
- ARM was developed by Acron Computer Limited of Cambridge, England between 1983 & 1985
- ARM limited founded in 1990
- ARM Cores
  - Licensed to partners to develop and fabricate new microcontrollers
  - Soft core

ARM Architecture

- Based upon RISC architecture with enhancement to meet requirements of embedded application
  - A large uniform register file
  - Load-Store architecture, where data processing operations operate on register content only
  - Uniform and fixed length instructions
  - 32-bit processor
  - Instructions are 32-bit long
  - Good Speed/ Power consumption ratio
  - High Code Density

Dr. V. Sathiesh Kumar Department of Electronics Engineering, MIT, India

Enhancement to basic RISC features

- Control over ALU and Shifter for every data processing operations to maximize their usage
- Auto-increment and auto-decrement addressing modes to optimize program loops
- Load and Store multiple instructions to maximize data throughput
- Conditional execution of instructions to maximize execution throughput

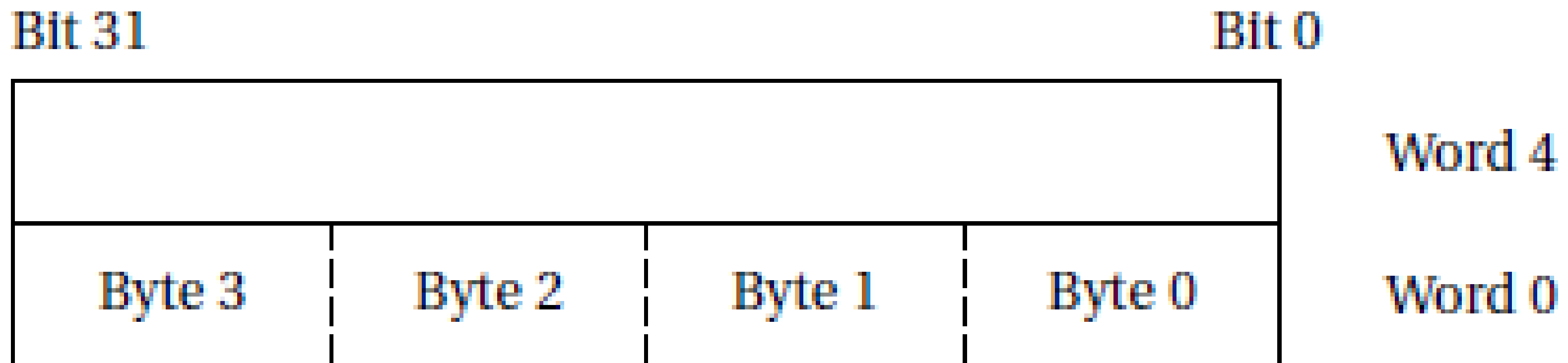
ARM Architecture Versions

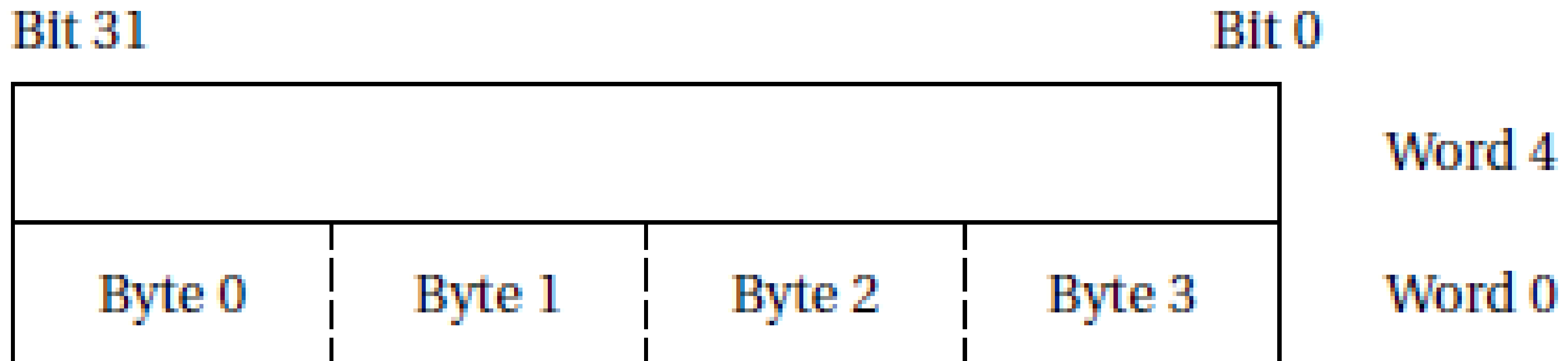
- Version 1 (1983-85): 26 bit addressing, no multiply or Co-processor
- Version 2: includes 32-bit result, multiply Co-processor
- Version 3: 32-bit addressing (Eg: ARM 6, ARM 7)
- Version 4: Add Signed, unsigned half word and signed byte load and store instructions (Eg: Strong ARM)
- Version 4T: 16-bit thumb compressed form of instruction introduced (Eg: ARM7TDMI)
- Version 5T: Superset of 4T adding new instructions
- Version 5TE: Add Signal processing signal extension (Eg: ARM 9E-S)

ARM Memory Organization

- PC is incremented by 4 to fetch the new instruction
- The ARM processor can be configured at power-up to address the bytes in a word in either little-endian mode (the lowest order byte residing in the low-order bits of the word) or big-endian mode (the lowest-order byte stored in the highest bits of the word)

ARM Memory Organization

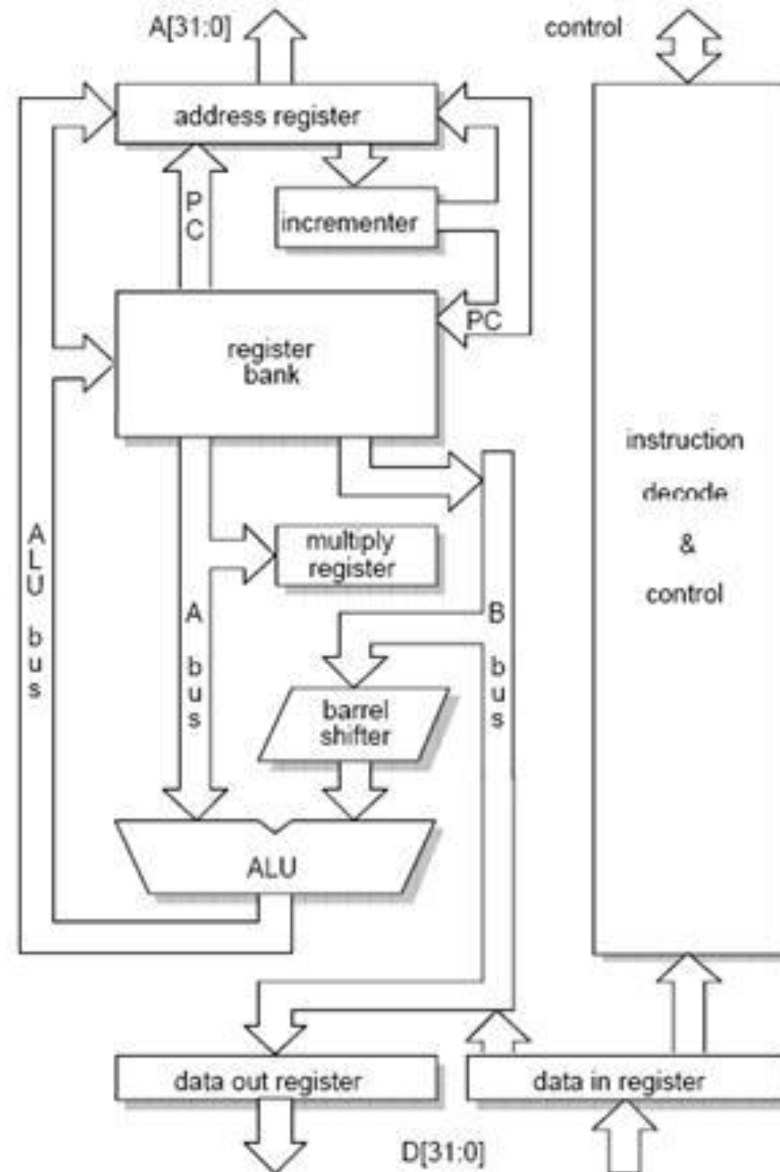




ARM Overview: Core Data Path

- Data items are placed in register file
  - No data processing instructions directly manipulate data in memory
- Instructions typically use two source registers and single result or destination registers.
- A barrel shifter on the data path can preprocess data before it enters ALU
  - Combinational circuit in which the shift (No of bits) takes place in a single clock cycle
- Increment/Decrement logic can update register content for sequential access independent of ALU
- 32-bit address and 32-bit data processing

Basic ARM Organization



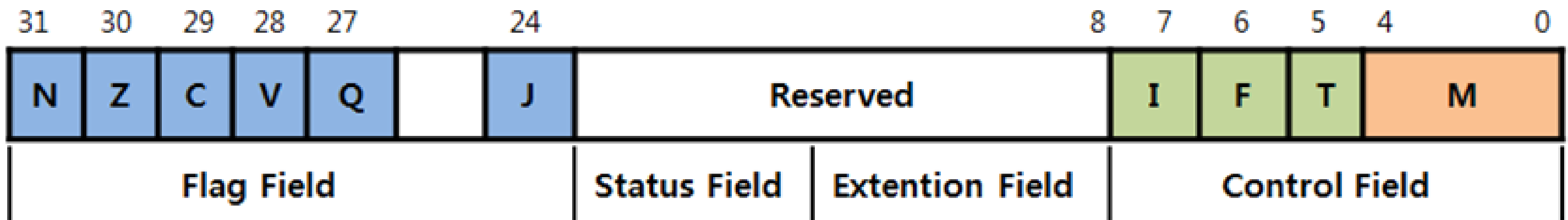
Registers

- The set of registers available for use by programs is called the programming model or programmer model. The CPU has many other registers that are used for internal operations and are unavailable to programmers
- General purpose registers (GPR) hold either data or address. All registers are of 32 bits. In User mode, 16 data registers and 2 status registers are visible
- Data registers: r0 to r15
  - r13: Stack Pointer
  - r14: Link Register (Where return address is put whenever a subroutine is called)
  - r15: Program counter

Registers- Continued

- Depending upon context, register r13 and r14 can also be used as GPR
- Any instruction which use r0 can as well be used with any other GPR (r1-r13)
- In addition, there are two status registers
  - CPSR: Current Program Status Register
  - SPSR: Saved Program Status Register
- Register r15 (Program counter)
  - All instructions are 32 bit wide and word aligned
  - PC value is stored in bits [31:2] with bits [1:0] undefined

CPSR



Flag Field	
N	Negative result from ALU
Z	Zero result from ALU
C	ALU operation caused Carry
V	ALU operation oVerflowed
Q	ALU operation saturated
J	Java Byte Code Execution

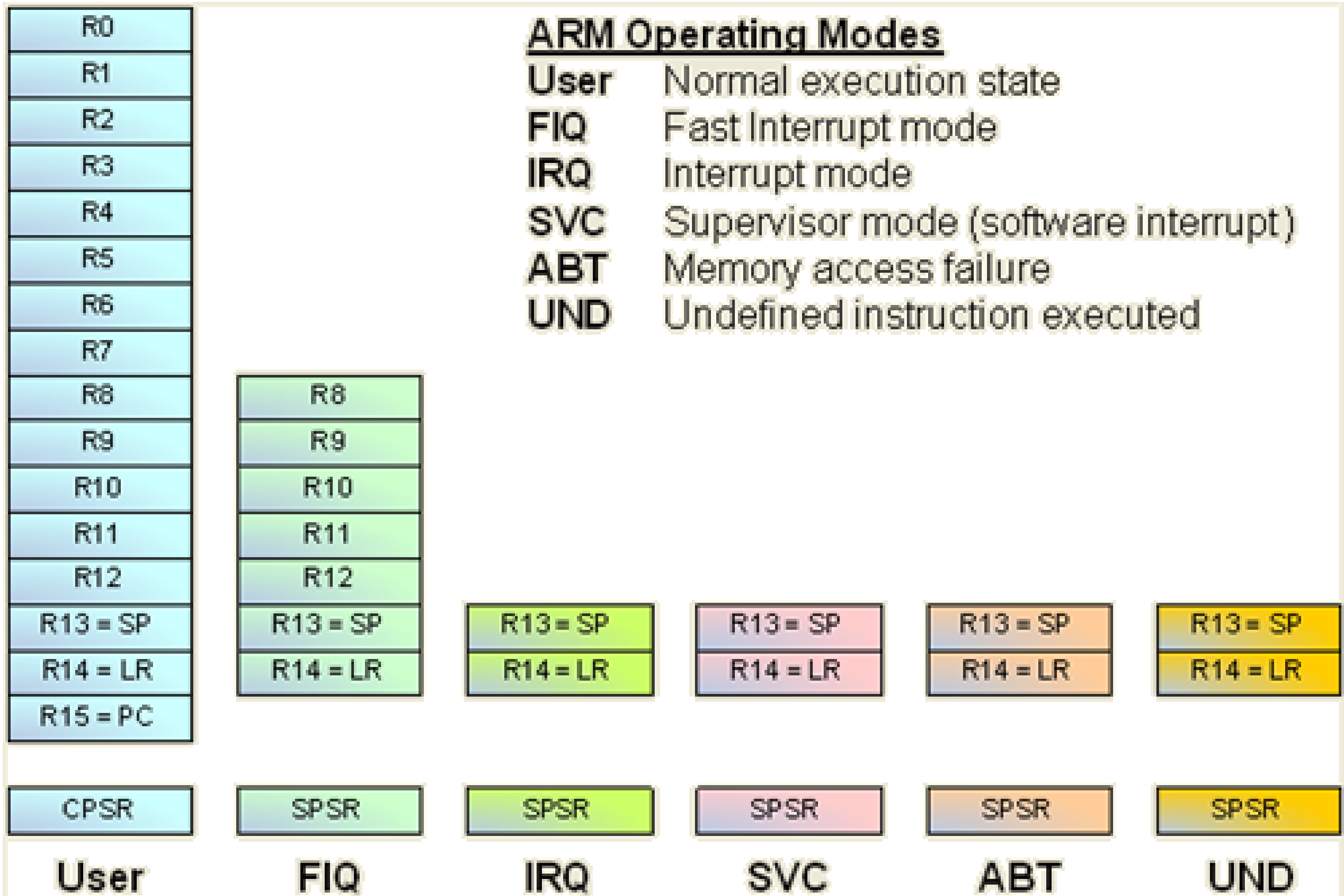
Control bits	
I	1: disables IRQ
F	1: disables FIQ
T	1: Thumb, 0: ARM

Mode bits M[4:0}	
0b10000	User
0b11111	System
0b10001	FIQ
0b10010	IRQ
0b10011	SVC(Supervisor)
0b10111	Abort
0b11011	Undefined

Banked Registers

- Register file contains in all 37 registers
  - 20 registers are hidden from program at different times. These registers are called banked registers
- Banked registers are available only when the processor is in a particular mode
  - Processor modes (other than System mode) have a set of associated banked registers that are a subset of 16 registers
  - Maps one to one onto a User mode register

Registers Banking-Currently visible in particular mode



Processor Modes

- Processor modes determine
  - which registers are active, and
  - access rights to CPSR register itself
- Each processor mode is either,
  - Privileged: Full read-write access to the CPSR
  - Non-Privileged: Only read access to the control field of CPSR but read-write access to the condition flags
- ARM has seven modes
  - Privileged: Abort, Fast interrupt request, Interrupt request, Supervisor, System and Undefined
  - Non-Privileged: User (Programs and application)

Privileged Processor Modes

- **Abort:** When there is a failed attempt to access memory
- **Fast interrupt request (FIQ) and Interrupt request (IR):** Correspond to interrupt levels available on ARM
- **Supervisor mode:** State after reset and generally the mode in which OS Kernel executes
- **System mode:** Special version of user mode that allows full read-write access of CPSR
- **Undefined:** When processor encounters an undefined instruction

Dr. V. Sathiesh Kumar Department of Electronics Engineering, MIT, India

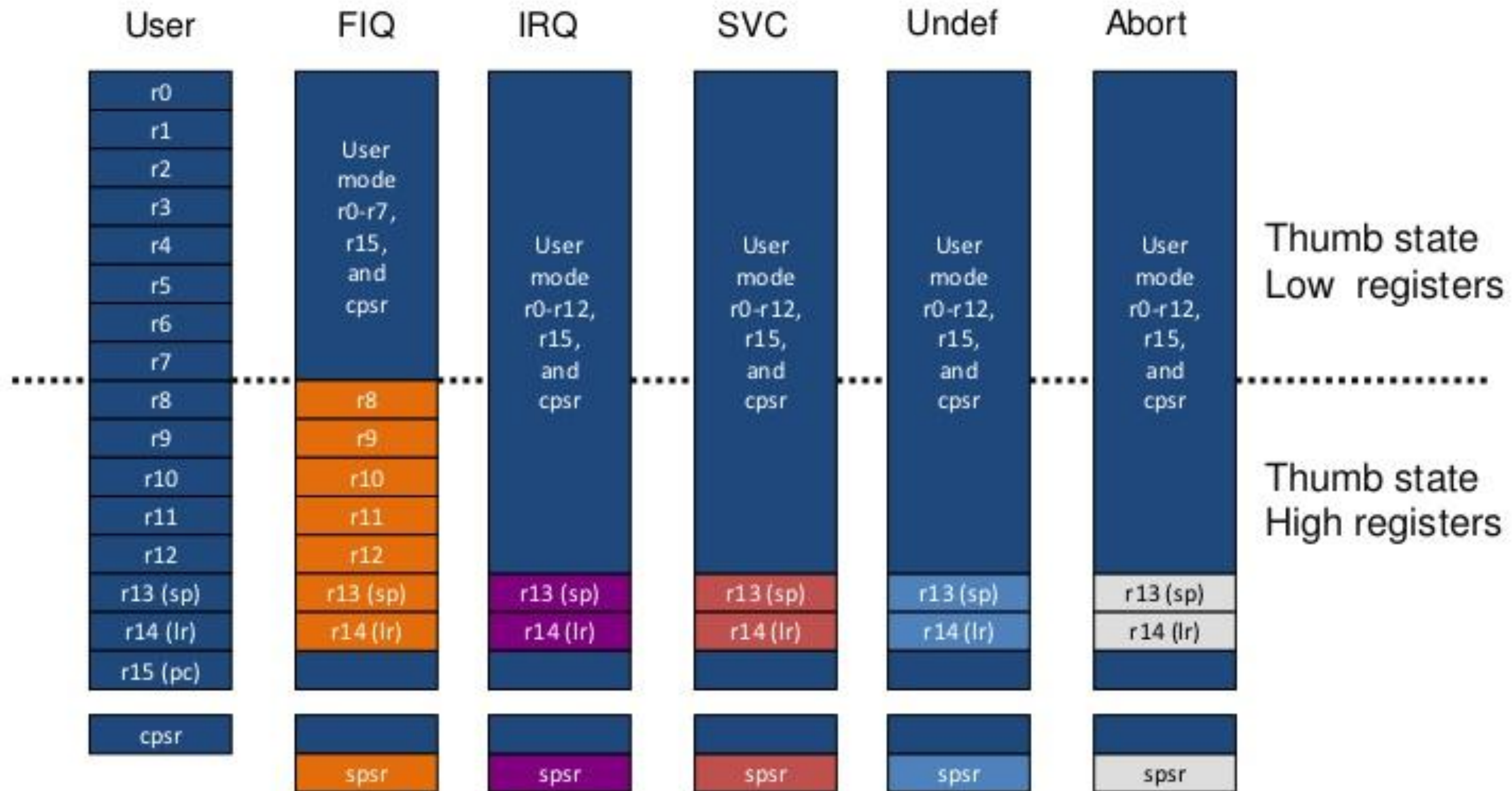
SPSR

- Each Privileged mode (except System mode) has associated with it a save program status register or SPSR
- This SPSR is used to save the state of CPSR when the Privileged mode is entered in order that the user state can be fully restored when the user process is resumed

Mode Changing

- Mode changes by writing directly to CPSR or by hardware when the processor responds to exception or interrupt
- To return to User mode a special return instruction is used that instructs the core to restore the original CPSR and banked registers

Register Organization in different modes



ARM Instruction set

- Instructions process data held in registers and access memory with load and store instructions.

- Classes of instructions:
  - Data processing
  - Branch instructions
  - Load-Store instructions
  - Software interrupt instructions
  - Program status register instructions

ARM Instruction set-Features

- Three address data processing instructions (two operands and destination)
- Condition execution of every instructions
- Load and store multiple registers
- Shift, ALU operation in a single instruction
- Open instruction set extension through the co-processor instruction

ARM Data types

- Word is 32-bit long
- Word can be divided into four 8-bit bytes
- ARM addresses can be 32-bit long
- Address refer to a byte
  - Address 4 starts at byte 4
- Can be configured at power-up as either little or big-endian mode

Data Processing

- Manipulate data within registers
  - MOVE instruction
  - Arithmetic instruction includes Multiply instruction
  - Logical instruction
  - Comparison instruction
- Suffix S on data processing instructions updates flags in CPSR
- Operands are 32-bit wide (registers or specified as literals)
- Second operand sent to ALU via a barrel shifter
- 32-bit result placed in register; long multiply instructions produces 64-bit result

MOVE Instruction

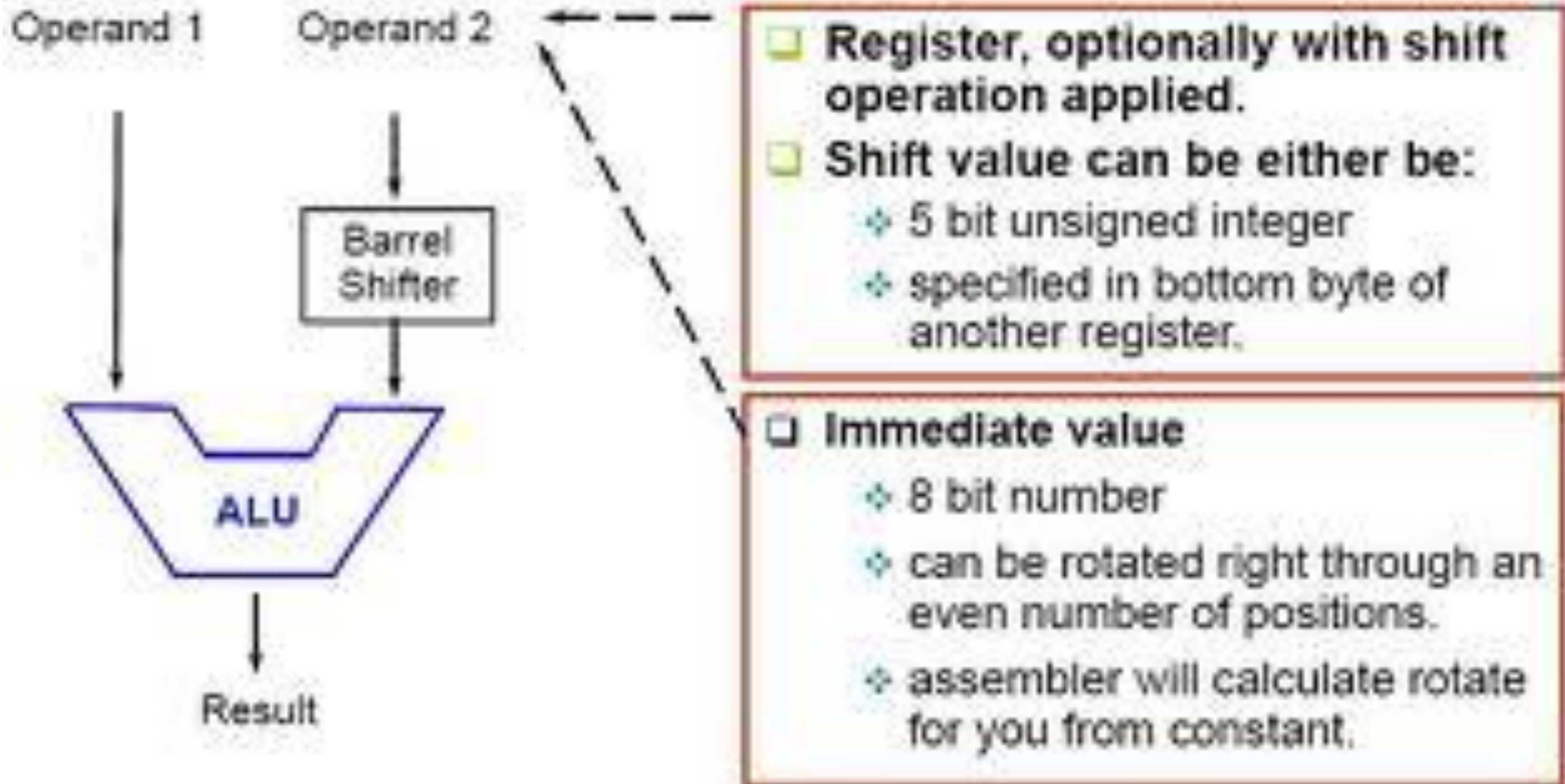
- MOV Rd, N (Eg: MOV r7, r5)

  Rd- Destination register

  N- can be an immediate value or source register

- MVN Rd, N (Move into Rd not of the 32-bit value from source)

Using Barrel Shifter



Using Barrel Shifter

- Enables shifting 32-bit operand in one of the source registers left or right by a specific number of positions within the cycle time of instructions
- Basic barrel shifter operations
  - Shift left, Shift right, Rotate right
- Facilitates fast multiply, division and increases code density
- Eg: MOV r7, r5, LSL #2 (Multiplies content of r5 by 4 and puts the result in r7)

LSL: Logical Shift Left

Arithmetic Instructions

- Implements 32 bits addition and subtraction.

  - 3 operand form

  Eg., SUB r0 , r1, r2

  - Subtract value stored in r2 from that of r1 and store in r0.

  Eg., SUBS r1, r1 , #1

  - Subtract 1 from r1 and store result in r1 and update z and c flags.

With Barrel Shifter

- Use of Barrel shifter with arithmetic and logical instructions increases the set of possible available operations.
  - Eg., ADD r 0, r1, r1, LSL #1
  - Register r1 is shifted to the left by 1, then it is added with r1 and the result (3 times r1) is stored in r0.

Multiply Instructions

- Multiply contents of a pair of registers.
  - Long multiply generates 64 bit result.

  Eg., MUL r0, r1 , r2 (32 bit result) r0=r1*r2

  - Contents of r1 and r2 multiplied and put in r0.

  Eg., UMULL r0 , r1 , r2, r3 (64 bit result) [r0, r1]=r2*r3

  - Unsigned long multiply with result stored in r0 and r1.

- No. of cycles taken for execution of multiply instruction depends upon processor implementation.

Dr. V. Sathiesh Kumar Department of Electronics Engineering, MIT, India

Multiply and Accumulate

- Result of multiplication can be accumulated with contents of another register.

Eg., MLA Rd , Rm , Rs , Rn (32 bit result) [Convolution Operation]

Rd = (Rn* Rs) + Rm

Eg., UMLAL Rdlo , Rdhi , Rm , Rn (64 bit result)

[Rdhi , Rdlo]= [Rdhi , Rdlo]+ (Rm* Rn)

Logical Instructions

- Bit wise logical operations on the two source registers.

  AND, OR, Ex-OR, bit clear

  Eg., BIC r0 ,r1 ,r2

  r2 contains a binary pattern where every binary 1 in r2 clears a corresponding bit location in register r1.

  Useful in manipulating status of flags and interrupt masks.

Compare Instructions

- Enables comparison of 32 bit values.

updates CPSR flags but do not affect other registers.

Eg., CMP r0 , r9

Flags set as a result of r0- r9.

Eg., TEQ r0 , r9

Flags set as a result of r0 Ex-OR r9.

Eg., TST r0 , r9

Flags set as a result of r0 AND r9.

Load-Store Instructions

- Transfer data between memory and process registers.
    - Single Register Transfer
    - Data types supported are signed and unsigned words (32 bits), half words, bytes.
    - Multiple Register Transfer
    - Transfer multiple registers between memory and the processor in a single instruction.
    - Swap
    - Swaps content of a memory location with the content of a register.

Single Transfer Instructions

- Load and store data on a boundary alignment.
  - LDR , LDRH , LDRB (Load- Word, Half word, Byte)
  - STR , STRH , STRB (Store – Word, Half word, Byte)
- Support different addressing modes
  - Register indirect: LDR r0, [r1]
  - Immediate: LDR r0, [r1, #4]
  - Register operation: LDR r0, [r1,-r2] (Address calculated using base register and another register)
  - Scaled (Address is calculated using the base address register and a barrel shifter operation)
  - Pre & Post indexing

Dr. V. Sathiesh Kumar Department of Electronics Engineering, MIT, India

Single Transfer Instructions-Pre & Post Indexing

- Pre-index with write back: LDR r0,[r1, #4]! (Updates the address base register with new address)
- Post-index: LDR r0, [r1], #4 (Updates the address register after address is used)
- Eg: Pre-indexing with write back: LDR r0, [r1, #4]!

Before instruction execution	After instruction execution
r0=0x00000000, r1=0x00009000	r0=0x02020202, r1=0x00009004
Mem 32[0x00009000]=0x01010101	
Mem 32[0x00009004]=0x02020202	

Multiple Register Transfer Instructions

- Load-Store multiple instructions transfers multiple register contents between memory and the processor in a single instruction
- More efficient- For moving blocks of memory and saving and restoring context and stack
- These instructions can increase interrupt latency (Usually instruction executions are not interrupted by ARM)

Multiple Byte Load-Store Instructions

- Any subset of current bank of registers can be transferred to memory or fetched from memory
- Eg: LDM SDM
- The base register Rn determines source or destination address

Addressing Modes

- LDMIA/IB/DA/DB   Ex: LDMIA Rn!, {r1-r3}
- STMIA/IB/DA/DB

		Start Address	End Address	Rn!
IA	Increment After	Rn	Rn+4*N-4	Rn+4*N
IB	Increment Before	Rn+4	Rn+4*N	Rn+4*N
DA	Decrement after	Rn-4*N+4	Rn	Rn-4*N
DB	Decrement before	Rn-4*N	Rn-4	Rn-4*N

N-register content   N=3 (r1 to r3)

Stack Processing

- A stack is implemented as a linear data structure which grows up (ascending) or down (descending)
- Stack pointer holds the address of the current top of the stack

Modes of Stack Operation

- ARM multiple register transfer instructions support
  - Full ascending: grows up, SP points to the highest address containing a valid item
  - Empty ascending: grows up, SP points to the first empty location above stack
  - Full descending: grows down, SP points to the lowest address containing a valid data
  - Empty descending: grows down, SP points to the first location below the stack

Some Stack Instructions

- Full ascending

  LDMFA: translates to LDMDA (POP)

  STMFA: translates to STMIB (PUSH)

  SP points to the last item in stack

- Empty descending

  LDMED: translates to LDMIB (POP)

  STMED: translates to STMIA (PUSH)

  SP points to first unused location

Swap Instruction

- Special case of load-store instruction

  SWP: Swap a word between memory and register

  SWPB: Swap a byte between memory and register

- Useful for implementing synchronization primitives likes semaphores

Control flow Instructions

- Branch instructions
- Conditional branches
- Conditional execution
- Branch and Link instructions
- Subroutine return instructions

Control flow-Branch Instructions

- **Branch instruction**: B label Ex: B forward

  Address label is stored in the instruction as a signed PC-relative offset

- **Conditional branch**: B <Cond> label Ex: BNE loop

  Branch has a condition associated with it and executed if condition codes have the current value Ex: Block memory copy

```
loop    LDMIA r9!, {r0-r7}
        STMIA r10!, {r0-r7}
        CMP r9, r11
        BNE loop
```

r9 points to source of data, r10 points to start of destination data, r11 points to end of the source

Control flow-Conditional Execution

- An unusual feature of ARM instruction set is that conditional execution applies not only to branches but to all arm instructions

  Ex: ADDEQ r0,r1,r2

  Instruction will only be executed when the zero flag is set to 1

- Advantages:
  - Reduces the number of branches
    - Reduces the number of pipeline flushes
    - Improves performance of the codes
  - Increases code density
  - Whenever the conditional sequence is 3 instructions or fewer (Smaller and faster) to exploit conditional execution than to use a branch

Control flow-Conditional Execution

Mnemonic	Flags required for execution	Description
GE	N == V	Greater than or equal to -Signed
GT	N == V & (Z Clear)	Greater Than -Signed
LT	N != V	Less Than -Signed
LE	(Z set) || (N != V)	Less Than -Signed
NE	Z clear	Not Equal to
HS	C set	Higher or equal -unsigned
HI	C set and Z clear	Higher than -unsigned
LS	(C clear) || (Z set)	Less than or equal to -unsigned
LO	C clear	Less than -unsigned
EQ	Z set	EQUAL
PL	N clear	Positive
CS	C set	Carry Set
CC	C Clear	Carry Clear
VS	V set	Overflow
VC	V clear	No Overflow
NV	N/A	NEVER -Probably best not to use
AL	N/A	Always

Branch and Link Instruction

- Perform a branch, save the return address following the branch in the link register (r14)

  Ex: BL Subroutine

- For nested subroutines, push r14 and some work registers required to be saved onto stack in memory

  Ex: BL Sub1

  STMFD r13!, (r0-r2, r14)

  BL Sub2

Subroutine Return Instruction

- No specific instruction

  Ex: Sub …

  MOV PC, r14

  Ex: When return address has been pushed to stack

  Sub2 …

  LDMFD r13!, (r0-r2, PC)

  Register transfer cannot be interrupted

Dr. V. Sathiesh Kumar Department of Electronics Engineering, MIT, India

Software Interrupt Instruction (SWI)

- A SWI causes a software interrupt exception, which provides a mechanism for applications to OS routines
- Instruction: SWI {<Cond>} SWI_Number
- When a processor executes an SWI instruction, it sets the program counter PC to the offset 0x8 in the vector table
- Instruction also forces the processor mode to SVC, which allows an operating system routine to execute
- Switching of mode is possible
- SWI is typically executed in User mode

Software Interrupt Instruction (SWI)- Continued

- Instruction forces processor mode to supervisor mode (SVC)- this allows an OS routine to be executed in privileged mode
- Each SWI has an associated SWI number which is used to represent a particular function call or feature
- Parameter passing- through registers; Return value is also passed using registers

Software Interrupt Instruction (SWI)- Continued

Ex: PRE: CPSR=nzcvqift USER (i-interrupt, t-thumb)

PC=0x0000A000

lr=0x003fffff Link register

r0=0x12

Instruction: 0x0000A000 SWI 0x123456

POST: CPSR=nzcvqift_SVC

SPSR=nzcvqift_USER

PC= 0x00008004

lr= 0x0000A004 (lr=r14_SVC)

r0= 0x12

Dr. V. Sathiesh Kumar Department of Electronics Engineering, MIT, India

Program Status Register Instructions

- Two instructions to control PSR directly
- MRS- transfers contents of either CPSR or SPSR into a register
- MSR- transfers contents of registers to CPSR or SPSR

Ex: Enabling IRQ interrupt

PRE: CPSR=nzcvqift_SVC (i is not set)

MRS r1, CPSR

BIS r1, r1, # 0x80

MSR CPSR, r1

POST: CPSR=nzcvqIft_SVC (i is set)

Co-processor Instructions

- Used to extend the instruction set
  - Used by cores with a co-processor
  - Co-processor specific operations   Ex: Memory management unit
- Syntax: Co-processor data processing

  CDP {<Cond>} Cp, Opcode1, Cd, Cn, Cm, {Opcode2}

  Cp- Co-processor number between p0 to p15

  Opcode field describes co-processor operation

  Cd, Cn, Cm- Co-processor registers
- Also Co-processor register transfer and memory transfer instructions

Dr. V. Sathiesh Kumar   Department of Electronics Engineering, MIT, India

Thumb Instructions

- Thumb encodes a subset of the 32-bit instruction set into a 16-bit subspace
- Thumb has higher performance than ARM on a processor with a 16-bit data bus
- Thumb has higher code density
  - For memory constrained embedded system
- Only low registers r0 to r7 is fully accessible
  - Higher registers are accessible with MOV, ADD, CMP instructions
- Barrel shift operations are separate instructions

Thumb Instructions- Continued

Ex: Code density

ARM divide

```
    MOV r3, #0
loop
    SUBS r0, r0, r1
    ADDGE r3,r3, #1
    BGE loop
    ADD r2, r0,r1
```

5 Instructions x 4 Bytes= 20 Bytes

Thumb divide

```
    MOV r3, #0
loop
    ADD r3, #1
    SUB r0,r1
    BGE loop
    SUB r3, #1
    ADD r2, r0, r1
```

6 Instructions x 2 Byte= 12 Bytes

ARM-Thumb Interworking

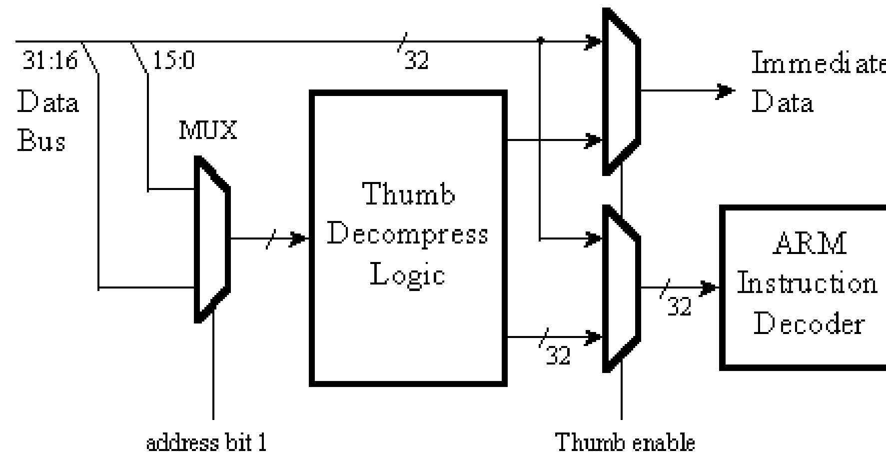
- To call a thumb routine from an ARM routine, the core has to change state (Changing T bit in CPSR)
- BX and BLX instruction can be used for the switch

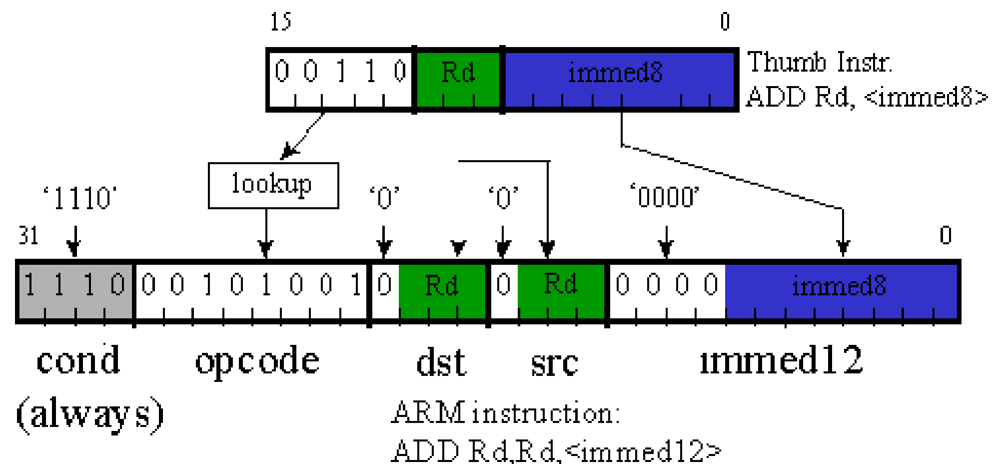
  Ex: BX r0; BLX r0

  Enters thumb state if bit 0 of the address in Rn is set to binary 1; otherwise it enters ARM state

Thumb (T) Architecture

- Thumb instruction decoder is placed in pipeline
- Change in thumb mode happens by changing the state of multiplexer A1
- A1 selects 16-bit data





ARM V5E Extensions

- Extensions to facilitate signal processing operations
- Supports
  - Signed multiply accumulate instructions
  - Saturation arithmetic
  - Creates flexibility and efficiency when manipulating 16-bit values for applications such as 16-bit digital audio processing

Saturation Arithmetic

- Normal ARM instructions wrap around when there is an overflow of an integer value
- Using ARM V5E instructions, you can saturate the result
  - Once the highest number is exceeded the result remains at the maximum value
  - Minimum value does not change an underflow

Ex: Instructions: QADD, QSUB