Chapter 15

ARM – Architecture, Programming and Development Tools

Lesson 4

ARM CPU–32 bit ARM Instruction set

Basic Programming Features-

- ARM code size small than other RISCs
- 32-bit un-segmented memory

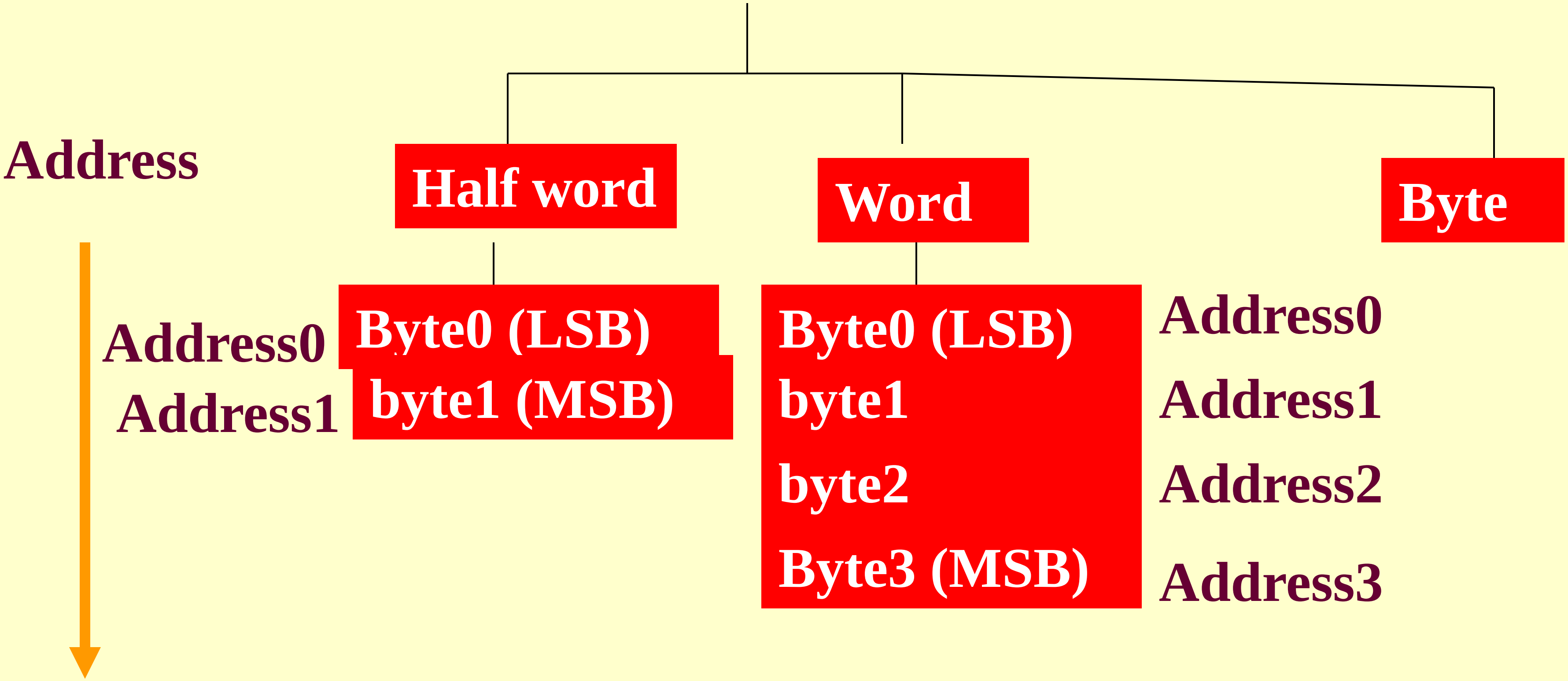
Basic Programming Features-

- Processor can operate on the words as per initialization.
- A word alignment can in big endian [least significant byte stored as higher bits (address 3) of a word] or little endian [least significant byte stored as lower bits (address 0) of a word].

Data Types support

- 8-bit byte,
- 16-bit half-word data types – half-words are aligned on 2-byte boundaries
- 32-bit data types- words are aligned on 4-byte boundaries

Little Endian Mode Word Data Types



Address

Half word

Word

Byte

Address0

Address1

Byte0 (LSB)

byte1 (MSB)

Byte0 (LSB)

byte1

byte2

Byte3 (MSB)

Address0

Address1

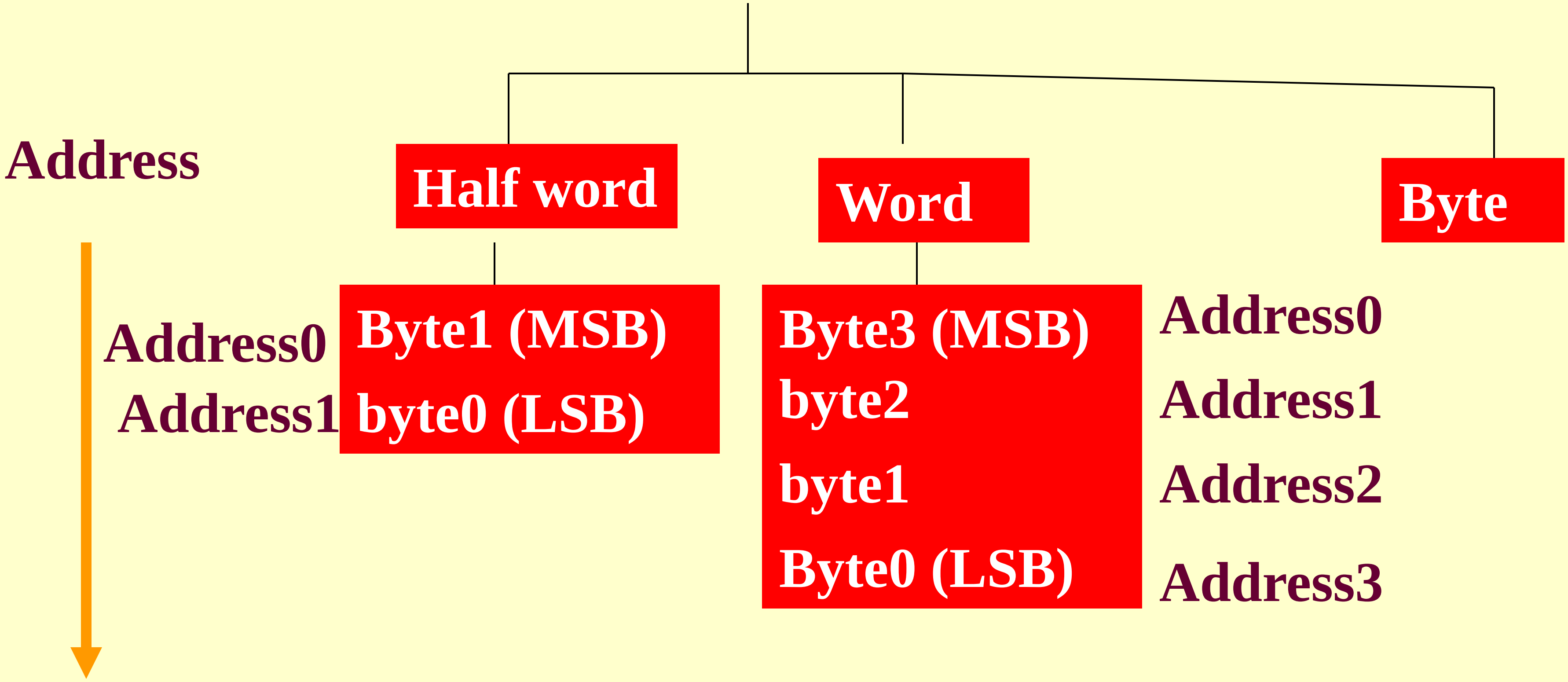
Address2

Address3

Address0- even address; Address1- odd address

Three DataType options

Big Endian Mode Word Data Types



Address

Half word

Word

Byte

Address0 Byte1 (MSB)

Address1 byte0 (LSB)

Byte3 (MSB) Address0

byte2 Address1

byte1 Address2

Byte0 (LSB) Address3

Address0- even address; Address1- odd address

Three DataType options

Programming Model

- 16 general-purpose registers with program counter as one of the register (R15).

- Registers

R0 R1 R2 R3 R4 R5 R6 R7 Lo registers

R8 R9 R10 R11 R12 Hi registers

SP (R13) Generally Stack Pointer

LR (R14) Link Register

PC (Rr15) Program Counter

CPSR SPSR

CPSR and SPSR

- CPSR (Conditions and Processor Status Register)
- SPSR (Saved Program Status Register) Saves Program Status Register from CPSR on branch and link (routine call) and SPSR can be stacked for each processor mode

Load Instructions

- LDM: Load multiple registers
- LDR: Load a register (32 bit)
- LDRB: Load a byte (8 bit) into register

Load Instructions (in certain variants e.g. ARM7500, StrongARM)

- LFM: Load multiple floating point values - ARM7500
- LDRSB: Load a signed byte (8 bit) into register
- Load a half-word (16 bit) into register (in certain variants e.g. StrongARM)

Store Instructions

- STM: Store multiple registers
- STR: Store a register (32 bit)
- STRB: Store a byte (8 bit) from a register

Store Instructions (in certain variants e.g. ARM7500, StrongARM)

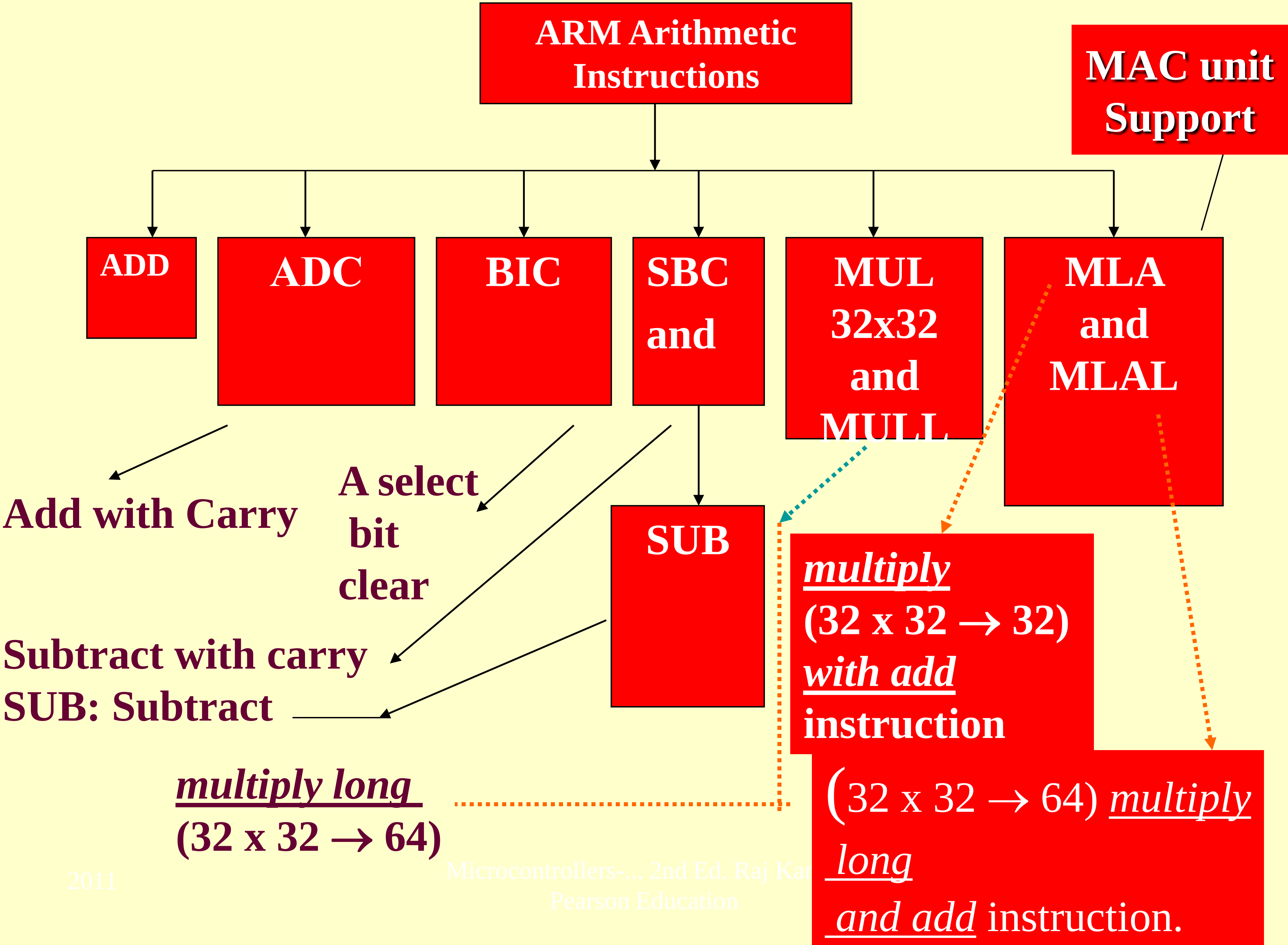
- SFM: Store Multiple Floating point values- ARM7500
- STRSB: Store a signed byte from a register- StrongARM
- STRSH: Store a signed half-word from a register- StrongARM

Load-store architecture **Features**

- LDM/STM with multiple options of addressing modes (like immediate)
- LDM/STM loads into a subset of registers in a list or save from the list to the memory addresses

Data Transfer Instructions

- MOV: Move value/register into a register
- MRC: Co-processor register transfer (co-processor to ARM)
- MVN: Move after negation



ARM Arithmetic Instructions

MAC unit Support

ADD

ADC

BIC

SBC and

MUL 32x32 and MULL

MLA and MLAL

Add with Carry

A select bit clear

SUB

Subtract with carry

SUB: Subtract

multiply (32 x 32 → 32) *with add* instruction

multiply long (32 x 32 → 64)

(32 x 32 → 64) *multiply long and add* instruction.

Arithmetic Instructions

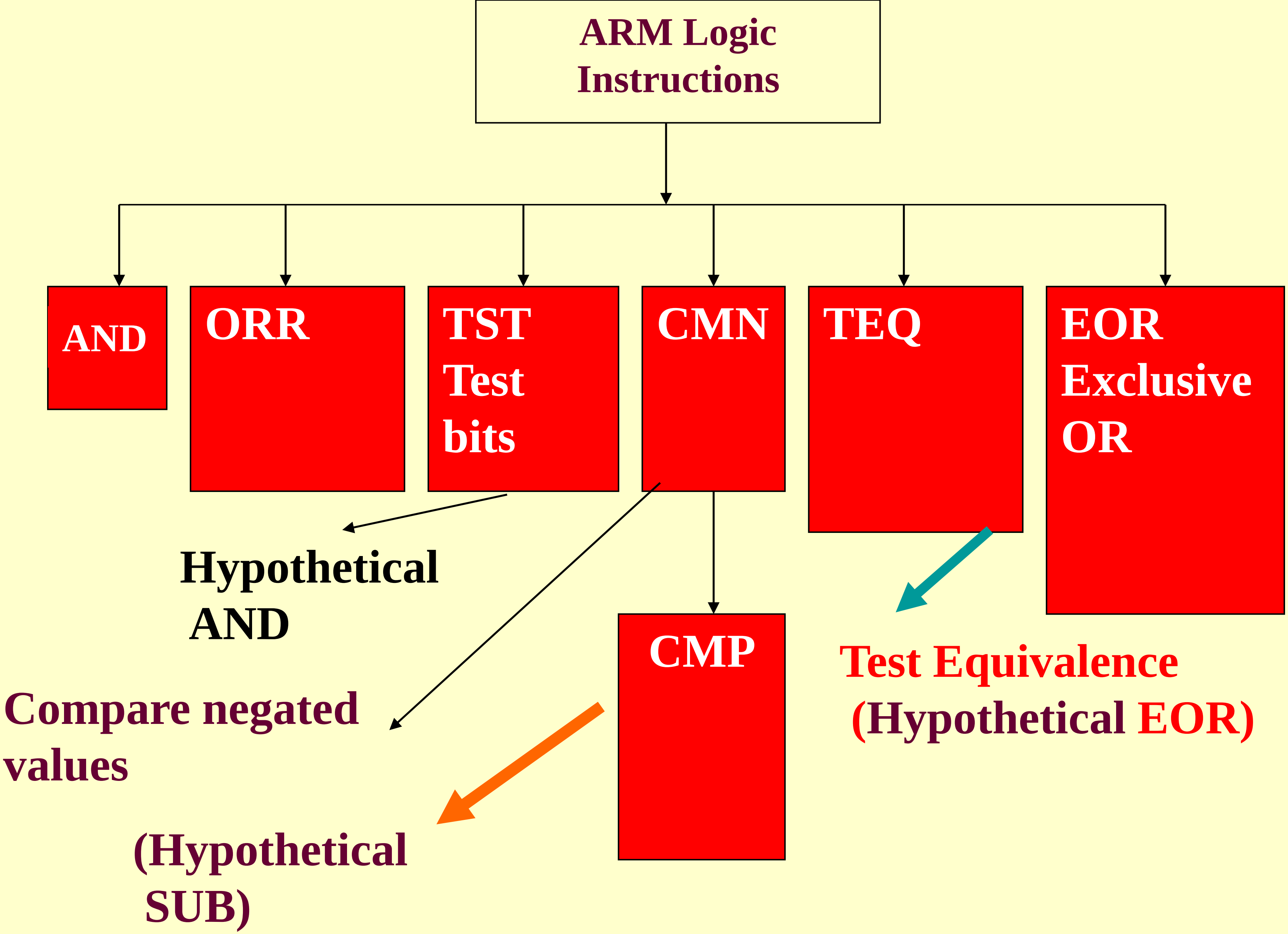
- ADC: Add with Carry, ADD:Add
- SBC: Subtract with carry , SUB: Subtract
- RSC: Reverse Subtract with carry , RSB: Reverse Subtract
- BIC: A select bit Clear

MUL and DIV Instructions..

- UMULL: Unsigned Long (64 bit) Multiply- StrongARM
- DVF: Divide floating Point- ARM7500

MAC unit Support

- MLA and MLAL implement DSP instructions
- UMLAL: Unsigned Long (64 bit) Multiply with Accumulate-Strong ARM



ARM Logic Instructions

AND

ORR

TST Test bits

CMN

TEQ

EOR Exclusive OR

CMP

Hypothetical AND

Compare negated values

(Hypothetical SUB)

Test Equivalence (Hypothetical EOR)

Logical Instructions..

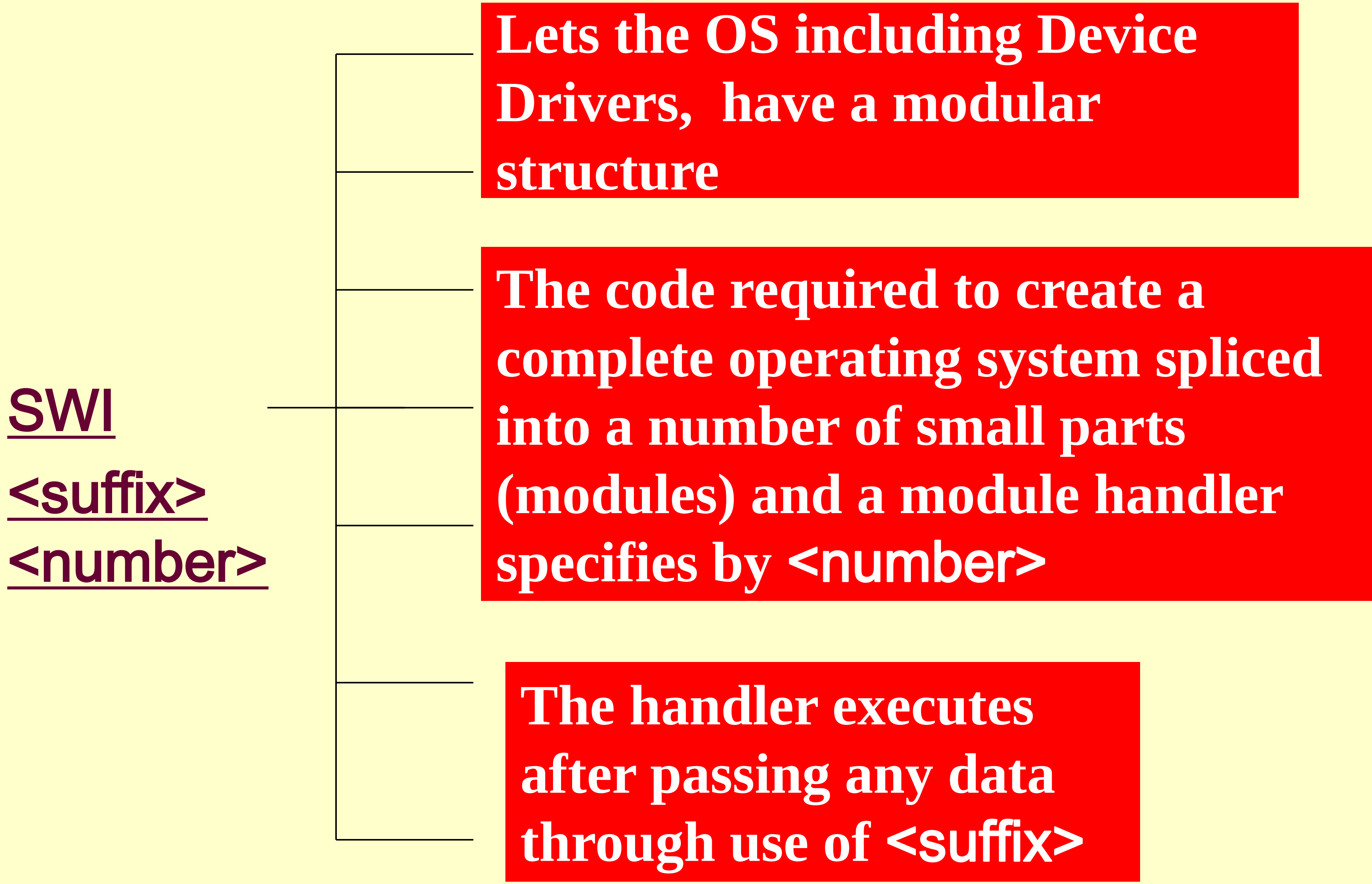
- Logical AND, ORR, EOR
- TST: Test bits (Hypothetical AND)
- TEQ: Test Equivalence (Hypothetical EOR)
- CMP: Compare values (Hypothetical SUB), CMN: Compare negated values

Program Flow Control …

- B: Branch
- BL: Branch with Link

Interrupt Control Instruction

- SWI<suffix> <number>
- SWI: cond SWI (8 msbs) remaining 24 bit lsbs are irrelevant as far as execution of the instruction is concerned.
- Interrupt handler can use the 24-bits for the suffix *parameter (s)* and number to reflect the interrupting foreground program (source of SWI).



SWI <suffix> <number>

Lets the OS including Device Drivers, have a modular structure

The code required to create a complete operating system spliced into a number of small parts (modules) and a module handler specifies by <number>

The handler executes after passing any data through use of <suffix>

ARM32-bit Instruction Set Features

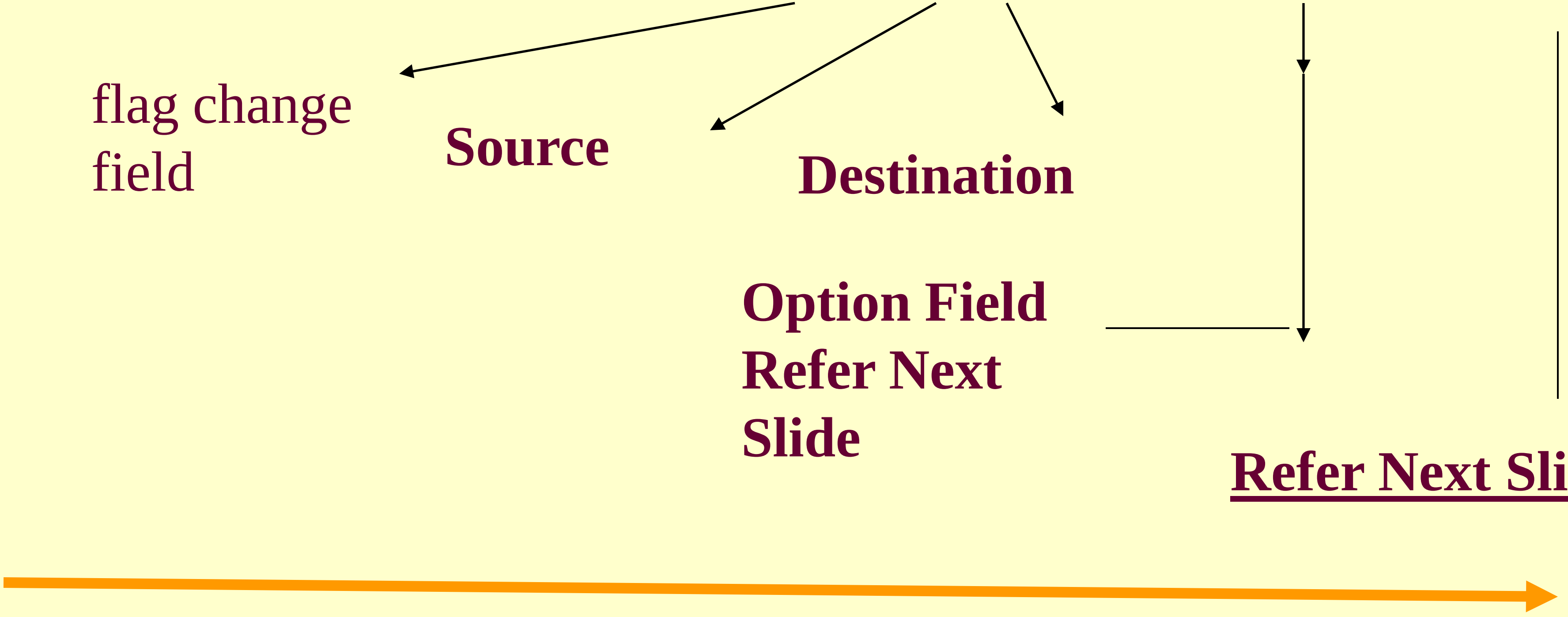
- Data processing, data transfer and control flow *execute* with several settable options
- Implementation of many instructions - by just a few instructions at the set
- CPU instruction decoder decodes instruction as per chosen options

ARM32-bit Instruction Option fields

- Conditional or no-condition 4-bit field inmost instructions - facilitates implementation of many conditional instructions
- After *cond* option, there are option fields like addressing mode, shift or rotate option, flag change fields

32-bit Instruction Format

Cond	Opcode major	X-Bit	Opcode Minor	S-Bit	Rn	Rd	4-Bits	operand 2
4-bits	2-bits		4-bits		4-bits	4-bits		8-bits



flag change field

Source

Destination

Option Field Refer Next Slide

Refer Next Slide

Bits 31-0

Instruction Format

- cond 00 X opcode S Rn Rd operand 2

S specifies CPSR flags change or no change by the operation

Instruction Format

- If X =1 then b11-b8 specifies rotate right instruction and b7-b0 immediate operand. b11-b8 = 0000 means no rotation, b11-b8 = 0001 means rotate right 2 times, b11-b8 = 1111 means ROR 30 times. Immediate operand is32-bit with 24 msbs = 0s

Instruction Format

- Second source operand can also be shifted or rotated before using the source in the operations.
- Flags (N, Z, C and V)can be changed or unchanged as per S bit in the instruction. Move can be conditional or unconditional as per *cond* 4-bit field in the instruction

Popular References at Web

http://www.cs.umd.edu/class/fall2001/cmsc411/proj01/arm/home.html

http://www.e-lab.de/ARM7/ARM-instructionset.pdf

http://www.cvsi.fau.edu/shankar/Presentations/Lecture7_Ch2_ARM.pdf http://www.cs.umd.edu/class/fall2001/cmsc411/proj01/arm/thumb.html

http://www.cs.man.ac.uk/Study_subweb/Ugrad/coursenotes/cs1031/Lec20-Thumb.pdf.

Summary

We learnt

- **Programmability as little endian and big endian**
- **ARM 7 - Princeton architecture, ARM 9 - Harvard architecture**
- 8-bit byte, 16-bit half-word and 32-bit data types–

We learnt

- Conditional data processing, data transfer and control flow Execution and addressing modes facilitate by the few **instructions only**
- Options for shift or rotate an operand in many instructions

We learnt

- LDM/STM with many more addressing modes(like immediate)
- MAC Support

End of Lesson 4 on ARM 32-bit