Universitatea de Vest din Timisoara, Facultatea de Matematica si Informatica

ARHITECTURA CALCULATOARELOR

Informatica, an I, 2021-2022

Dr. Mafteiu-Scai Liviu Octavian

Universitatea de Vest din Timisoara, Facultatea de Matematica si Informatica
ARHITECTURA CALCULATOARELOR, Informatica, an I, 2021-2022

CURS 11+

1. Limbajul de asamblare pentru procesoare ARM partea a 2-a

Dr. Mafteiu-Scai Liviu Octavian

ARM PROCESSOR VS. INTEL PROCESSOR

INTEL - CISC (Complex Instruction Set Computing) :

- more operations than ARM-
- more addressing modes than ARM
- less registers than ARM

ARM – RISC (Reduced instruction set Computing) :

- a simplified instruction set (100 instructions or less)
- more general purpose registers than CISC
- ARM uses instructions that operate only on registers
- uses a Load/Store memory model for memory access (only Load/Store instructions can access the memory)

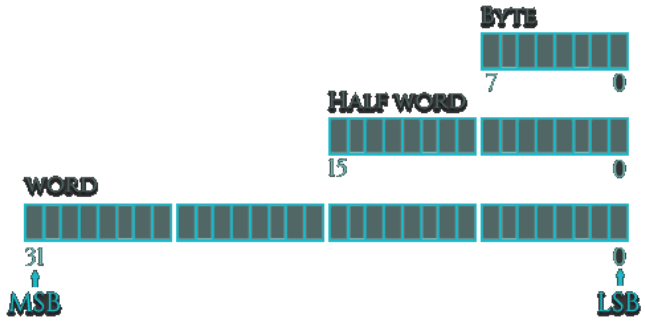
  example: incrementing a 32-bit value at a particular memory address require three types of instructions: load, increment and store (load the value at a particular address into a register, increment it within the register, and store it back to the memory from the register)

ARM versions

ARM family	ARM architecture
ARM7	ARM v4
ARM9	ARM v5
ARM11	ARM v6
Cortex-A	ARM v7-A
Cortex-R	ARM v7-R
Cortex-M	ARM v7-M

Data types

- Bytes
- Halfwords
- Words



- Singed data types
- Unsigned data types

Examples:

LDR = Load Word

LDRH = Load unsigned Half Word

LDRSH = Load signed Half Word

LDRB = Load unsigned Byte

LDRSB = Load signed Bytes

STR = Store Word

STRH = Store unsigned Half Word

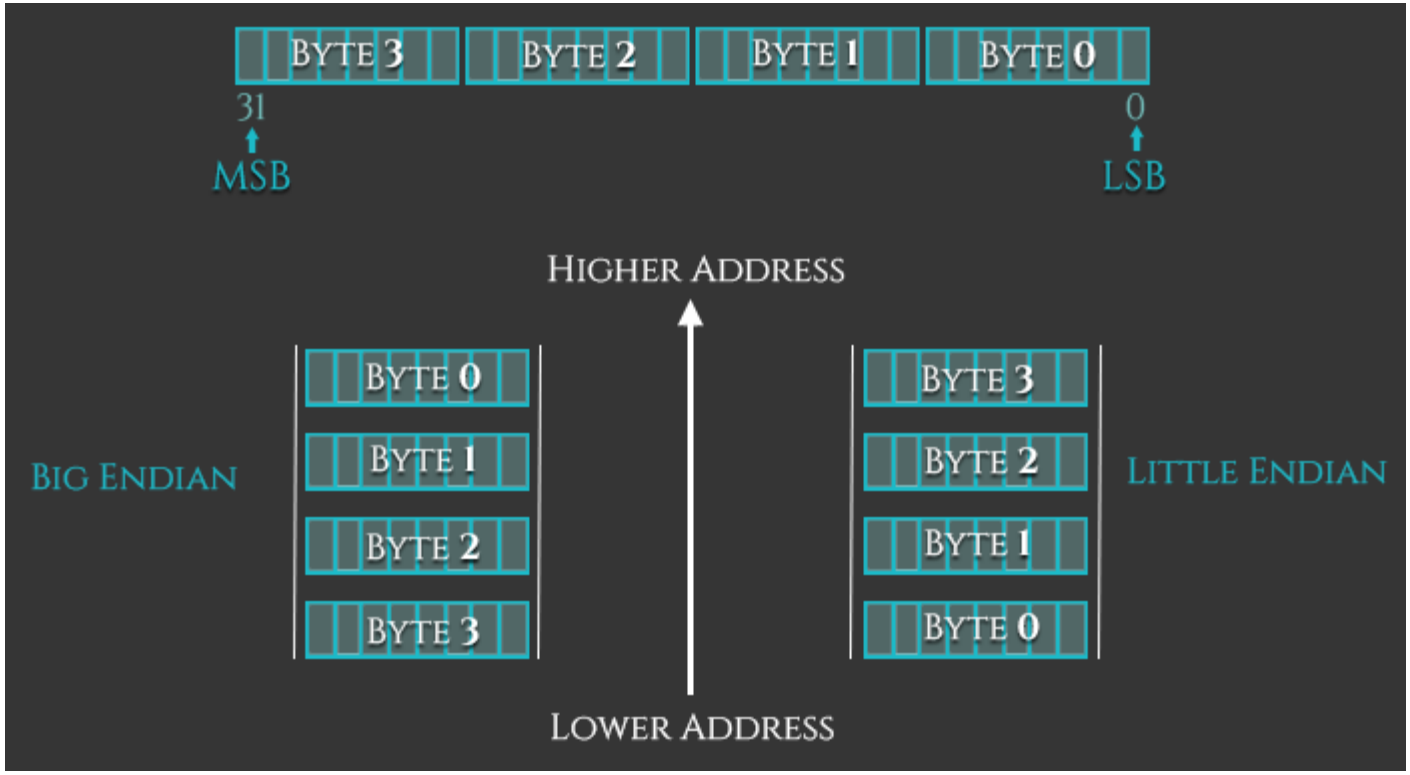
STRSH = Store signed Half Word

STRB = Store unsigned Byte

STRSB = Store signed Byte

Ways of viewing bytes in memory

- Little-Endian (LE) (the **least**-significant-byte ist stored at the lowest address)
- Big-Endian (BE) (the **most**-significant-byte is stored at the lowest address.)



ARM registers

30 general registers (32 bit)

- the first 16 registers are accessible in user-level mode (r0-15)
- the additional registers are available in privileged software execution

#	Alias	Purpose
R0	–	General purpose
R1	–	General purpose
R2	–	General purpose
R3	–	General purpose
R4	–	General purpose
R5	–	General purpose
R6	–	General purpose
R7	–	Holds Syscall Number
R8	–	General purpose
R9	–	General purpose
R10	–	General purpose
R11	FP	Frame Pointer

Special Purpose Registers

R12	IP	Intra Procedural Call
R13	SP	Stack Pointer
R14	LR	Link Register
R15	PC	Program Counter
CPSR	–	Current Program Status Register

ARM registers – using

R0-R12: common operations to store temporary values, pointers (locations to memory), etc.

R0, for example, can be referred as accumulator during the arithmetic operations or for storing the result of a previously called function.

R7 is useful while working with syscalls as it stores the syscall number

! the function calling convention on ARM specifies that the first four arguments of a function are stored in the registers r0-r3.

R13: SP (Stack Pointer). The Stack Pointer points to the top of the stack. R13 is used for allocating space on the stack

R14: LR (Link Register). When a function call is made, the Link Register gets updated with a memory address referencing the next instruction where the function was initiated from. Doing this allows the program return to the “parent” function that initiated the “child” function call after the “child” function is finished.

R15: PC (Program Counter). The Program Counter is automatically incremented by the size of the instruction executed.

ARM instructions syntax

MNEMONIC{S}{condition} {Rd}, Operand1, Operand2

MNEMONIC - Short name (mnemonic) of the instruction

{S} - An optional suffix. If S is specified, the condition flags are updated on the result of the operation

{condition} - Condition that is needed to be met in order for the instruction to be executed

{Rd} - Register (destination) for storing the result of the instruction

Operand1 - First operand. Either a register or an immediate value

Operand2 – Second (flexible) operand. Can be an immediate value (number) or a register with an optional shift

Operand2 is called a flexible operand, because we can use it in various forms as immediate value (with limited set of values), register or register with a shift.

Example:

#123 - Immediate value (with limited set of values).

Rx - Register x (like R1, R2, R3 ...) Rx,

ASR n - Register x with arithmetic shift right by n bits (1 = n = 32) Rx,

LSL n - Register x with logical shift left by n bits (0 = n = 31) Rx,

LSR n - Register x with logical shift right by n bits (1 = n = 32) Rx,

ROR n - Register x with rotate right by n bits (1 = n = 31) Rx,

RRX - Register x with rotate right by one bit, with extend

Examples:

ADD R0, R1, R2 - adds contents of R1 (Operand1) and R2 (Operand2 in a form of register) and stores the result into R0 (Rd)

ADD R0, R1, #2 - adds contents of R1 (Operand1) and the value 2 (Operand2 in a form of an immediate value) and stores the result into R0 (Rd)

MOVLE R0, #5 - moves number 5 (Operand2, because the compiler treats it as MOVLE R0, R0, #5) to R0 (Rd) ONLY if the condition LE (Less Than or Equal) is satisfied

MOV R0, R1, LSL #1 - moves the contents of R1 (Operand2 in a form of register with logical shift left) shifted left by one bit to R0 (Rd). So if R1 had value 2, it gets shifted left by one bit and becomes 4. 4 is then moved to R0.

The most common instruction on ARM

Instruction	Description
MOV	Move data
MVN	Move 2's complement
ADD	Addition
SUB	Subtraction
MUL	Multiplication
LSL	Logical Shift Left
LSR	Logical Shift Right
ASR	Arithmetic Shift Right
ROR	Rotate Right
CMP	Compare
AND	Bitwise AND
ORR	Bitwise OR

Instruction	Description
EOR	Bitwise XOR
LDR	Load
STR	Store
LDM	Load Multiple
STM	Store Multiple
PUSH	Push on Stack
POP	Pop off Stack
B	Branch
BL	Branch with Link
BX	Branch and eXchange
BLX	Branch with Link and eXchange
SWI/SVC	System Call

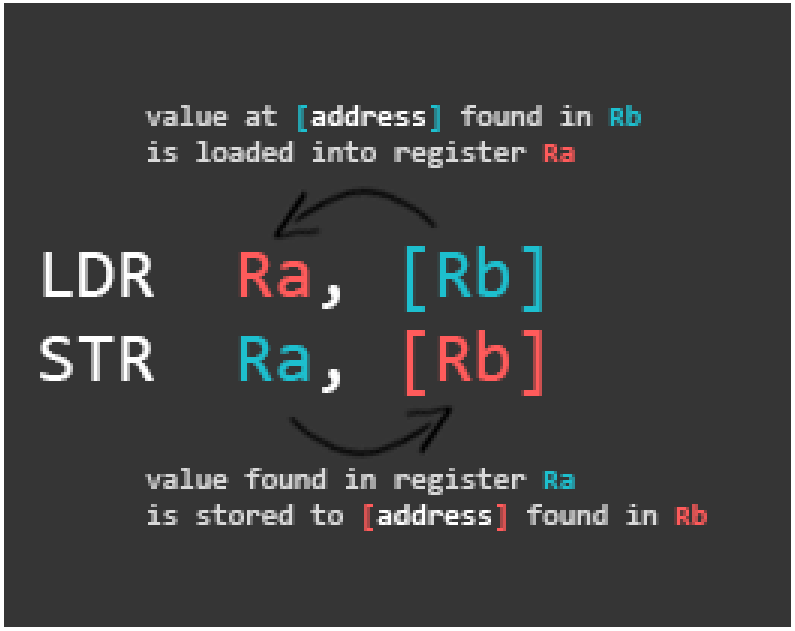
Memory instructions- LOAD and STORE

LDR is used to load something from memory into a register

STR is used to store something from a register to a memory address.

LDR R2, [R0] @ [R0] - origin address is the value found in R0.
STR R2, [R1] @ [R1] - destination address is the value found in R1.

LDR operation: loads the value at the address found in R0 to the destination register R2.
STR operation: stores the value found in R2 to the memory address found in R1.



LOAD and STORE multiple values at once

LDM load multiple

STM store multiple

LDM r0, {r4,r5}

load two word values from the memory pointed by R0

! loaded multiple (2 data blocks) with one command

STM r1, {r4,r5}

store multiple values to memory

takes values from registers R4 and R5 and stores these values to a memory location specified by R1

PUSH and POP instructions

Work on stack

PUSH to load from the Stack

POP to store onto the Stack

The Stack Pointer (SP) is a register which will always point to an address wihin the Stack's memory region

PUSH: the address in SP gets DECREASED by 4.

POP: the address in SP gets INCREASED by 4.