Assembly Language Lecture Series: Software Architecture: RISC-V memory

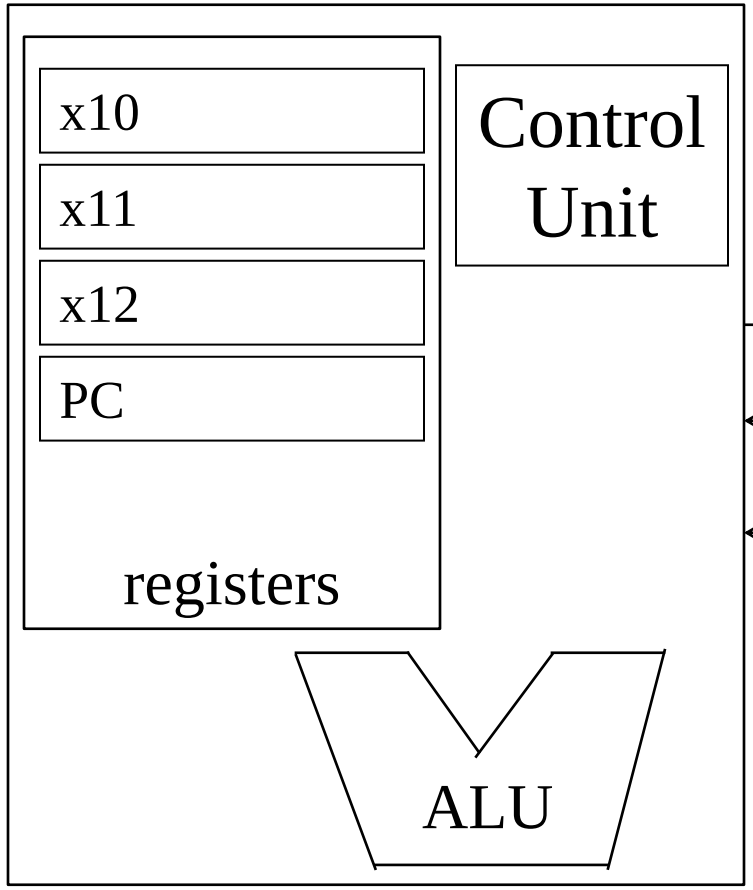
Sensei RL Uy
College of Computer Studies
De La Salle University
Manila, Philippines

Copyright Notice

Software architecture

Software Architecture: view the CPU in terms of software point of view

- Internal Registers + External Memory



RISC-V

address	Memory data (byte)
C	11111111
B	11111110
A	11001010
9	10101101
8	00001011
7	11110000
6	11011110
5	10111100
4	10011010
3	01111000
2	01010110
1	00110100
0	00010010

memory

Software architecture: memory

- Each memory location is *n*-bit in size (usually 8-bit [*byte addressable*])
- Accessing the memory requires distinct names [*label* or *variable name*] or addresses
- 2^k address constitutes the address space of the computer (i.e., from 0 to 2^k-1 successive locations in the memory)
- Data can be multi-byte (16-bit half*word*; 32-bit *word*; 64-bit *doubleword*
- **Endianness** refers to the convention used to interpret the byte ordering of multi-byte stored in the computer memory.
- Two types: **Big** endian and **Little** endian

Label /var	Address (hex)	Memory data (byte)(hex)
NANCY	C	FF
MAMA	B	FE
KILO	A	CA
JULIET	9	AD
INDIA	8	0B
HOTEL	7	F0
GAMMA	6	DE
FOXTROT	5	BC
ECHO	4	9A
DELTA	3	78
CHARLIE	2	56
BETA	1	34
ALPHA	0	12

memory

Software architecture: memory

Little Endian….

- The address of a datum is the address of the least-significant byte (LSB)
- Example: What is a 16-bit (2-byte data) found in address 2?
  78 56 (LSB)
- 78 56 (0x7856)
- Example: What is a 32-bit (4-byte data) found in address 8?
- 0xFECAAD0B

address	Memory data (byte)
C	FF
B	FE
A	CA
9	AD
8	0B
7	F0
6	DE
5	BC
4	9A
3	78
2	56
1	34
0	12

memory

Software architecture: memory

Big Endian….

- The address of a datum is the address of the most-significant byte (MSB)
- Example: What is a 16-bit (2-byte data) found in address 2?
  56 (MSB) 78
- 56 78 (0x5678)
- Example: What is a 32-bit (4-byte data) found in address 8?
- 0x0BADCAFE

address	Memory data (byte)
C	FF
B	FE
A	CA
9	AD
8	0B
7	F0
6	DE
5	BC
4	9A
3	78
2	56
1	34
0	12

memory

Software architecture: memory

- Memory alignment: access to memory larger than a byte must be *aligned*
- Problems with misalign memory: requires multiple aligned memory references
- Aligned if: data of size *s* bytes at byte address A is A mod *s* == 0
- Thus for 16-bit data, address is aligned at 0x0, 0x2,0x4,0x8, etc.
- Thus for 32-bit data address is aligned at 0x0, 0x04, 0x8, 0xC, 0x10, etc.
- Thus for 64-bit data, address is aligned at 0x0, 0x8, 0x10, 0x18, etc.

address	Memory data (byte)
C	FF
B	FE
A	CA
9	AD
8	0B
7	F0
6	DE
5	BC
4	9A
3	78
2	56
1	34
0	12

memory

Software architecture: memory

Software architecture and RISC-V architecture:

- RISC-V view each memory location as one-byte (i.e., byte addressable)
- RISC-V uses little endian ordering system

address	Memory data (byte)
C	FF
B	FE
A	CA
9	AD
8	0B
7	F0
6	DE
5	BC
4	9A
3	78
2	56
1	34
0	12

memory

Variable Declaration

Directives	Description
.byte	Declare initialized data as 8-bit integer
.half	Declare initialized data as 16-bit integer
.word	Declare initialized data as 32-bit integer
.dword	Declare initialized data as 64-bit integer
.float	Declare initialized data as (32-bit) single-precision floating point
.double	Declare initialized data as (64-bit) double-precision floating point
.asciz	Declare initialized data as string with null terminator

These type of variables are declare in:
.data

Example:

```
.data
 var1:  .byte  0x12
 var2:  .half  0x1234
 var3:  .word  0x12345678
 var4:  .float  4.5
 var5:  .asciz  “ABCD”
```

Variable Declaration

```
.data
 var1: .byte 0x12, 0x34, 0x56, 0x78
```

label	address	Memory data (byte)
	3	78
	2	56
	1	34
var1	0	12

```
.data
 var2: .half 0x1234, 0x5678
```

label	address	Memory data (byte)
	3	56
	2	78
	1	12
var2	0	34

```
.data
 var3: .word 0x12345678
```

label	address	Memory data (byte)
	3	12
	2	34
	1	56
var3	0	78

Load instruction

LW *rd*, *offset*(*rs*)

- LW instruction loads a 32-bit value from memory to *rd*.
- Memory address is obtained by adding register *rs* to the sign-extended 12-bit *offset*.
- pseudo-instruction *la* (load address) is used to initialize a register to point to a memory.

```
.data
var1:.word 0xBBAA9988
var2:.word 0x81ABCDEF
var3:.word 0xCCDDEEFF
var4:.word 0x000000FF
.text
la x5, var2
lw x10, 0(x5)
lw x11, 4(x5)
lw x12, -4(x5)
```

x10 = 81ABCDEF

x11 = ?

x12 = ?

x5 = ?

Label	Address (hex)	Memory data (hex)
	1001000F	00
	1001000E	00
	1001000D	00
var4	1001000C	FF
	1001000B	CC
	1001000A	DD
	10010009	EE
var3	10010008	FF
	10010007	81
	10010006	AB
	10010005	CD
var2	10010004	EF
	10010003	BB
	10010002	AA
	10010001	99
var1	10010000	88

Store instruction

SW *rs2*, *offset*(*rs1*)

- SW instruction stores the 32-bit value of register *rs*2 to the memory.
- memory address is obtained by adding register *rs1* to the sign-extended 12-bit *offset*.
- pseudo-instruction *la* (load address) is used to initialize a register to point to a memory.

```
.data
var1:.word 0xBBAA9988
var2:.word 0x81ABCDEF
var3:.word 0xCCDDEEFF
var4:.word 0x000000FF
.text
la x5, var1
la x6, var2
lw x10, 0(x5)
sw x10, 0(x6)
```

x10 = BBAA9988

x5 = ?

x6 = ?

Label	Address (hex)	Memory data (hex)
	1001000F	00
	1001000E	00
	1001000D	00
var4	1001000C	FF
	1001000B	CC
	1001000A	DD
	10010009	EE
var3	10010008	FF
	10010007	~~81~~ BB
	10010006	~~AB~~ AA
	10010005	~~CD~~ 99
var2	10010004	~~EF~~ 88
	10010003	BB
	10010002	AA
	10010001	99
var1	10010000	88