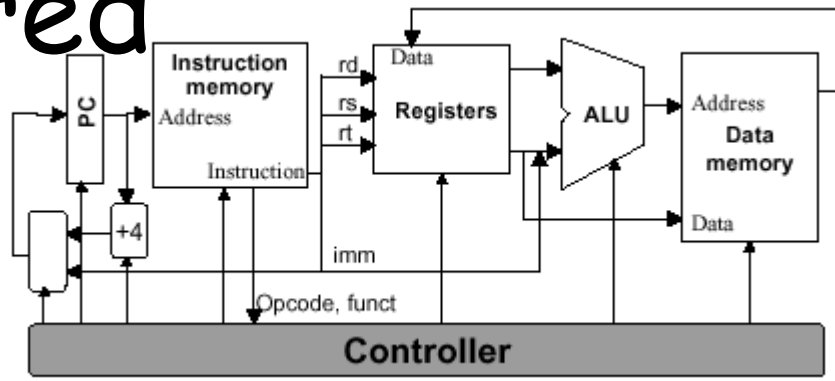
Lecture 8

28-01-2021

Variables are allocated to Registers…

- Registers are numbered from 0 to 31



- Each register can be referred to by either its number or name…
- Register Number referencing:

$0, $1, $2, … $30, $31

Registers Referred by Name

- Example:

  $16 - $23 → $s0 - $s7

  (correspond to variables)

  $8 - $15 → $t0 - $t7

  (correspond to temporary variables)

  Next will explain other 16 register names

- Referring by names usually make code more readable

MIPS Registers

Name	Register Number	Usage	Preserved by callee
$zero	0	hardwired 0	N/A
$v0-$v1	2-3	return value	no
$a0-$a3	4-7	arguments	no
$t0-$t7	8-15	temporary values	no
$s0-$s7	16-23	saved values	YES
$t8-$t9	24-25	more temporary values	no
$gp	28	global pointer	YES
$sp	29	stack pointer	YES
$fp	30	frame pointer	YES
$ra	31	return address	YES

MIPS Instruction Set

- Only **Load** instruction can read an operand from memory
- Only **Store** instruction can write an operand to memory
- **Typical RISC calculations:**
  - Load(s) to get operands from memory into registers
  - Calculations performed only on values in registers
  - Store(s) to write result from register back to memory

MIPS Assembly Language Examples

- **ADD $1,$2,$3**
  - Register 1 = Register 2 + Register 3
- **SUB $1,$2,$3**
  - Register 1 = Register 2 - Register 3
- **AND $1,$2,$3**
  - Register 1 = Register 2 AND Register 3
- **ADDI $1,$2,10**
  - Register 1 = Register 2 + 10
- **SLL $1, $2, 10**
  - Register 1 = Register 2 shifted left 10 bits

MIPS Assembly Language Examples

- **LW $1,100**
  - Register 1 = Contents of memory location 100
  - Used to load registers
- **SW $1,100**
  - Contents of memory location 100 = Register 1
  - Used to save registers
- **LUI $1,100**
  - Register 1 upper 16-bits = 100
  - Lower 16-bits are set to all 0's
  - Used to load a constant in the upper 16-bits
  - Other constants in instructions are only 16-bits, so this instruction is used when a constant larger than 16-bits is required for the operation

Few MIPS Assembly Language Examples

- **J 100**
  - Jump to PC+100
- **JAL 100**
  - Save PC in $31 and Jump to PC+100
  - Used for subroutine calls
- **JR $31**
  - Jump to address in register 31
  - Used for subroutine returns

MIPS Assembly Language Examples

- **BEQ $1, $2, 100**
  - If register 1 equal to register 2 jump to PC+100
  - Used for Assembly Language If statement
- **BNE $1, $2, 100**
  - If register 1 value **not equal** to register 2 jump to PC+100
  - Used for Assembly Language If statements

MIPS Labels

- Instead of absolute memory addresses:
  - **Symbolic labels are used to indicate memory addresses in assembly language**
- Assembly Language Programs are easier to modify and understand when labels are used…
- **Example:**

```
Loop: lw  R1, 1004
      lw  R2, 1008
      add R3, R1, R2
      sw  R3, 1000
      J Loop
```

  - Location 250 to be used for looping --- Use label LOOP instead of jump address 250

First MIPS Program:
A = B + C;

```
LW  $2, B    ;Register 2 = value at B

LW  $3, C    ;Register 3 = value at C

ADD $4, $2, $3 ;Register 4 = B+C

SW  $4, A    ; A = B + C
```

The Constant Zero

- $R0 (or $zero) is always contains constant 0
  - Cannot store any other value. What is the use?
- Useful for synthesizing common operations
  - E.g., move between registers

add $t2, $s1, $zero

  - **There is no separate Mov instruction**
  - **add R3,R0,3** #move 3 to R3
- **R0 helps in reducing the number of instructions.**

MIPS: Addressing modes

- Only immediate and displacement modes supported, besides register mode.
- Register indirect achieved by placing 0 in displacement field.
  - LW R4, 0(R1) //[R4]<--[[R1]]
- Absolute addressing achieved by using R0 as the base register.
  - LW R1, 100(R0) //[R1]<- [100]

MIPS Arithmetic

- All arithmetic instructions have 3 operands
- Operand order is fixed (destination first)
- Example

C code: **A = B + C**

Assume A,B,C in $s0, $s1, and $s2

MIPS code: **add $s0, $s1, $s2**

Temporary Variables

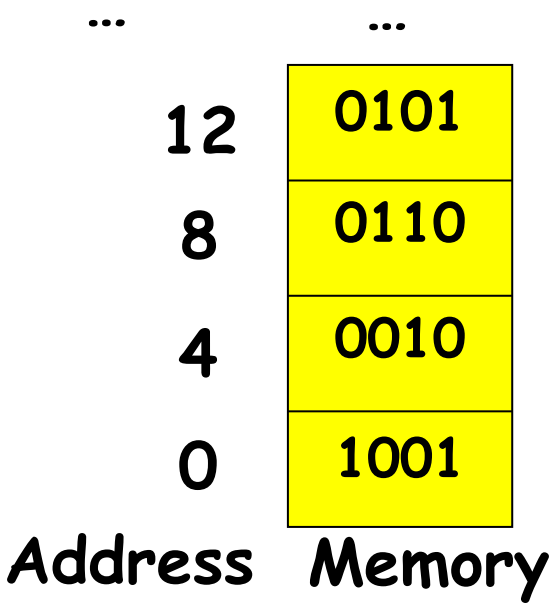
- Intermediate results of expressions are stored in temporary variables.

- C code: **f = (g + h) - (i + j);**

Write MIPS code: Assume f, g, h, i, j are in $s0 through $s4 respectively

Data Transfer

```
lw $t0, 8($s3)        # base address in $s3, offset 8
addi  $t0,$t0,100
sw $t0, 48($s3)       # store word
```



Byte – 8 bits

Word – 32 bits

Memory access in words

Address to byte level

2^{32} bytes with byte addresses from 0 to 2^{32}

2^{30} words with byte addresses 0, 4, 8, ... $2^{32}-4$

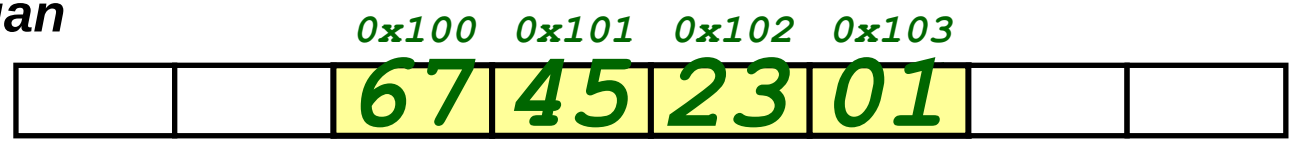
what are the least 2 significant bits of a word address?

Byte Ordering

- **Big Endian**
  - Least significant byte has highest address
- **Little Endian**
  - Least significant byte has lowest address
- **Example:**
  - Variable **v** has Hex representation **0x01234567**
  - Address given by **&v** is **0x100**

Big Endian

Little Endian



Exercise

- Consider **0xFF00AA11**

1000	11
1001	AA
1002	00
1003	FF

Little Endian

1000	FF
1001	00
1002	AA
1003	11

Big Endian

1. The address for a group of bytes is the smallest address of the four.
2. What goes in that byte is:
    1. Big Endian: the big end
    2. Little Endian: the little end
3. The remaining three bytes are filled in sequence.

Memory Operand Example 1

- C code:

```
g = h + A[8];
```

  - g in $s1, h in $s2, base address of A in $s3

- Compiled MIPS code:

  - Index 8 requires offset of 32
    - 4 bytes per word

```
lw  $t0, 32($s3)    # load word
add $s1, $s2, $t0
```