CSSE 232
Computer Architecture I

I/O and Addressing

Class Status

Reading for today

- 2.9-2.10, 6.6 (optional)

Outline

- I/O
- More memory instructions
- Addressing modes
- Jump and branch instructions

Input and Output

- How the processor communicates with the outside world
- Input
  - Keyboard
  - Mouse
  - Network card
- Output
  - Display / Graphics card
  - Printer
  - Network card
  - Sound card

Performing I/O

- I/O port
  - I/O devices are assigned to an I/O port
  - Use specific I/O instructions to access I/O ports
- Memory-mapped I/O
  - Reserve a portion of memory space for device I/O
  - Use regular memory instructions to access

I/O ports

- I/O devices are assigned a port ID
- CPU has special instructions to read and write to I/O ports
- Data can then be sent and received by devices

Memory Mapped I/O

- Designate a portion of memory space to be for I/O
- Device memory and registers are mapped to this space
- Use normal memory instructions to read and write data
- A special piece of hardware monitors memory operations
- If regular memory space is accessed
  - Read and write as normal
- If I/O space is accessed
  - Intercept memory read/write
  - Redirect to actual device memory
- I/O mapping can be disabled and addresses revert to regular memory

Polling and Interrupts

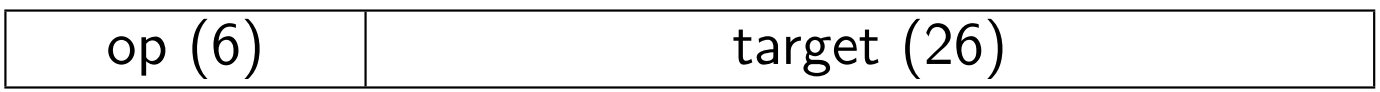
- **Polling**
  - Devices set flag indicating they need to send/receive data
  - Processor frequently polls devices
  - Checks if any I/O devices need attention, responds
- **Interrupt**
  - Device set interrupt flag and special status register
  - Processor is forced to stop executing normal code
  - Jumps to special interrupt handling code
  - Responds to device interrupt

More Memory Instructions

- MIPS can work with other memory sizes: bytes, shorts
- `lb` : load byte into lowest register byte
- `sb` : store lowest register byte into memory
- `lh` : load half into bottom register half
- `sh` : store lower register half into memory
- `ld` : load double word
- `sd` : store double word
- Upper bits are sign extended

J-Type

- Last instruction type
- Used for jumps
  - Jump only needs an address
  - No other values



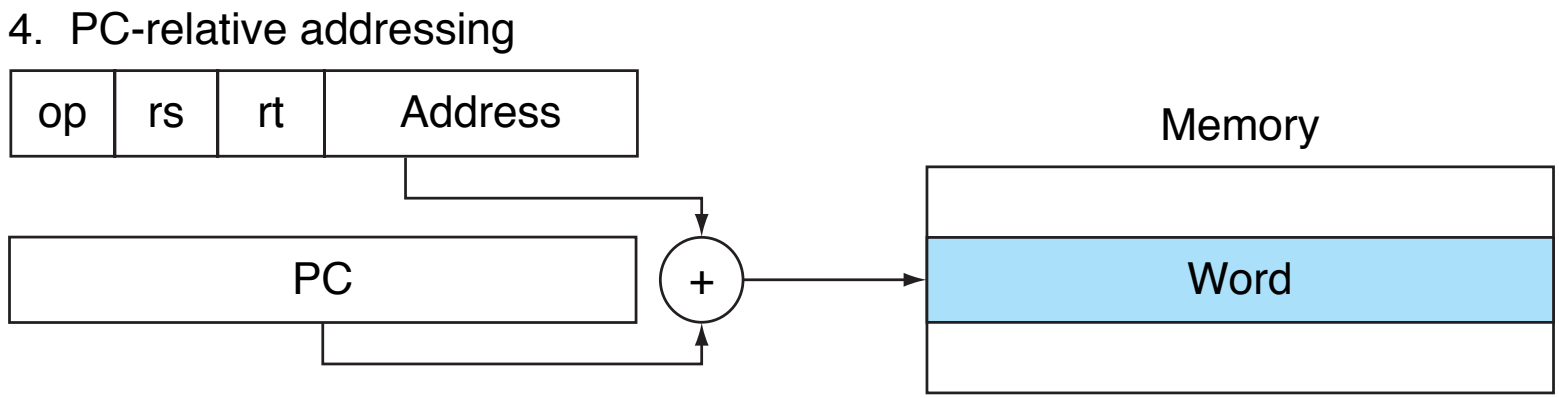
Addressing Modes

Related instructions

- ori
- addi
- jr
- lw
- sw
- beq
- bne
- j

PC-Relative Addressing

- Branch address calculated as
  1. Sign extend instruction constant
  2. Shift left 2 times (multiply by 4)
  3. Sum result with PC (actually PC+4)
- Allows branching within $\pm 2^{15}$ instructions
- Example: beq, bne



Program Counter (PC)

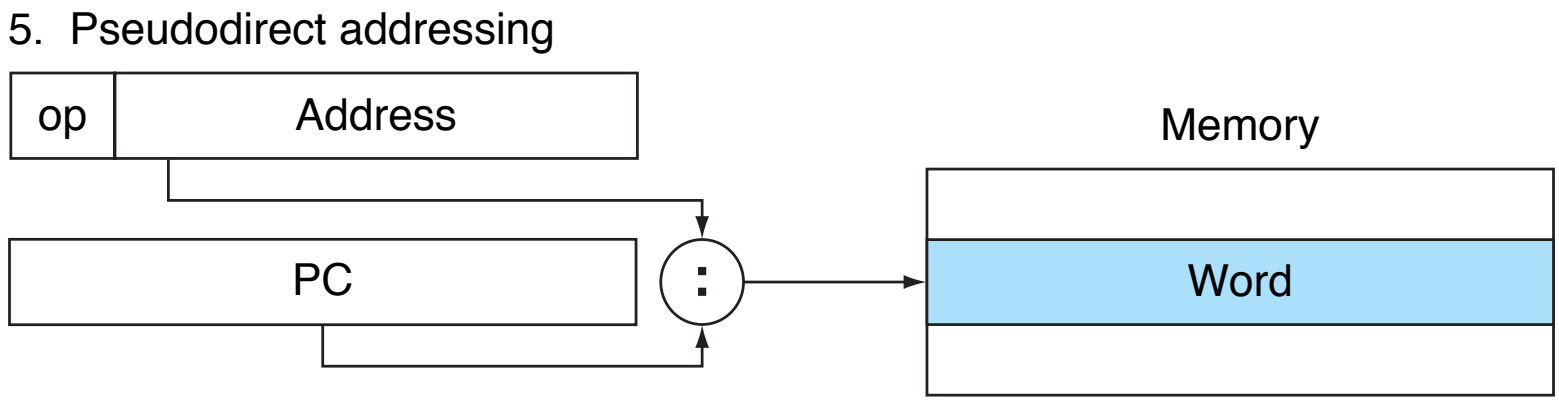
- Special register that points to current instruction
- `jal` saves PC+4 into `$ra`

Direct Addressing (also called Register)

- Register value allows access to 2^{32} locations/values
- Example: jr

Pseudo-Direct Addressing

- Jump address is calculated as
  1. Take 26 bits from instruction constant
  2. Shift left 2 times (multiply by 4)
  3. Concat top 4 bits of the PC as MSBs
- Allows branching to 2^{26} instructions
- Example: j



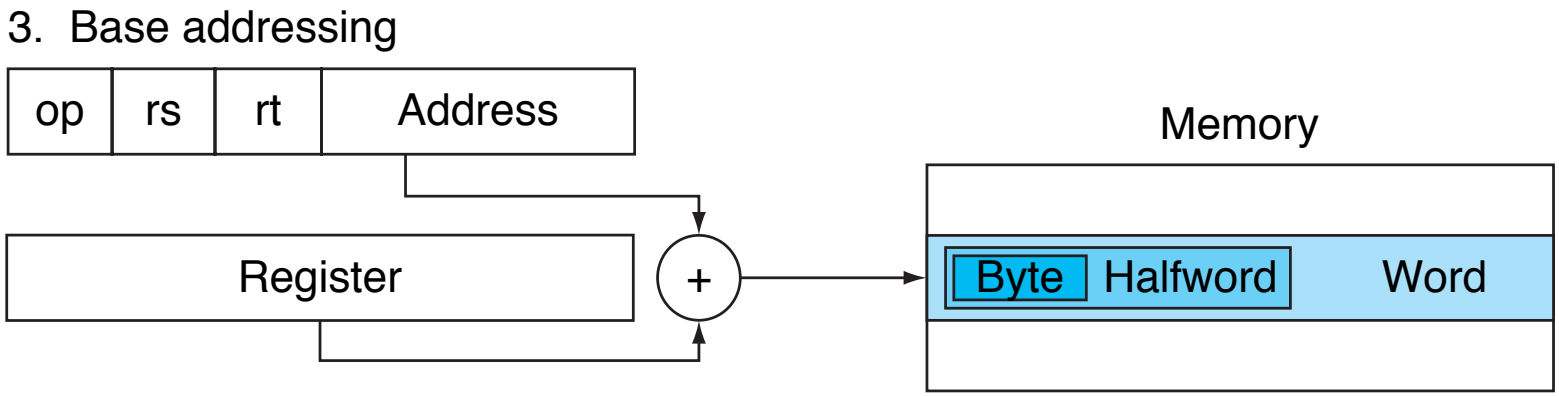
Immediate Addressing

- Operand is a constant within the instruction
- Some instructions sign extend (addi, slti)
- Some zero-extend (ori, andi, sltiu)
- Allows access to values in range $[0, 2^{16})$ or $[-2^{15}, 2^{15})$

1. Immediate addressing

Base / Displacement Addressing

- Memory address calculated as
  1. Sign extend instruction constant
  2. Add to register value
- Register allows access to 2^{32} locations
- Constant allows access to $\pm 2^{15}$ locations relative to base
- Require word (4 byte) alignment: lw, sw



Example

What would be the machine language equivalent (in decimal) of the following code? Assume the Loop code starts on location 80000 in memory.

```
Loop: sll  $t1, $s3, 2
      add  $t1, $t1, $s6
      lw   $t0, 0($t1)
      bne  $t0, $s5, Exit
      addi $s3, $s3, 1
      j    Loop
Exit:
```

Example

addr	instruction format					
80000	0	0	19	9	2	0
80004	0	9	22	9	0	32
80008	35	9	8	0		
80012	5	8	21	2		
80016	8	19	19	1		
80020	2	20000				

Branching Far Away

- What would you change if the `bne` statement needed to branch to 2^{21} instructions before or after the branch?
- What would you change if the `bne` statement needed to branch to 2^{30} instructions before or after the branch?

Branching Far Away

- What would you change if the `bne` statement needed to branch to 2^{21} instructions before or after the branch?
- What would you change if the `bne` statement needed to branch to 2^{30} instructions before or after the branch?
  - Have the `bne` go to a `beq`, resulting in up $2 * 2^{15}$ distance
  - Have the `bne` go to a `j`, for a distance of $2^{15} + 2^{26}$
  - Load the goal address in a register and have the `bne` go to a `jr`

Review and Questions

- I/O
- More memory instructions
- Addressing modes
- Jump and branch instructions