EECE 321: Computer Organization

Mohammad M. Mansour
Dept. of Electrical and Compute Engineering
American University of Beirut

Lecture 11: MIPS Instruction Formats

Announcements

Branch Example

opcode	rs	rt	immediate
6b	5b	5b	16b

- MIPS Code:

```
Loop:   beq   $9, $0, End
        add   $8, $8, $10
        addi  $9, $9, - 1
        j     Loop
End:
```

- beq branch is I-Format:
  - opcode = 4 (look up in table)
  - rs = 9 (first operand)
  - rt = 0 (second operand)
  - immediate = ???
- Immediate Field:
  - Number of instructions to add to (or subtract from) the PC, starting at the instruction following the branch.
  - In beq case, immediate = 3

decimal

opcode	rs	rt	immediate
4	9	0	3

binary

opcode	rs	rt	immediate
000100	01001	00000	0000 0000 0000 0011

Questions on PC-Relative Addressing

- Does the value in branch field change if we move the code?
- What do we do if destination is > 2^{15} instructions away from branch?
- Since it's limited to ± 2^{15} instructions, doesn't this generate lots of extra MIPS instructions?
- Why do we need all these addressing modes? Why not just one?

J-Format Instructions

- For branches, we assumed that we don't want to branch too far, so we can specify change in PC.
- For general jumps (j and jal), we may jump to anywhere in memory.
- Ideally, we could specify a 32-bit memory address to jump to.
- Unfortunately, we can't fit both a 6-bit opcode and a 32-bit address into a single 32-bit word, so we compromise.
- Define "fields" as follows:

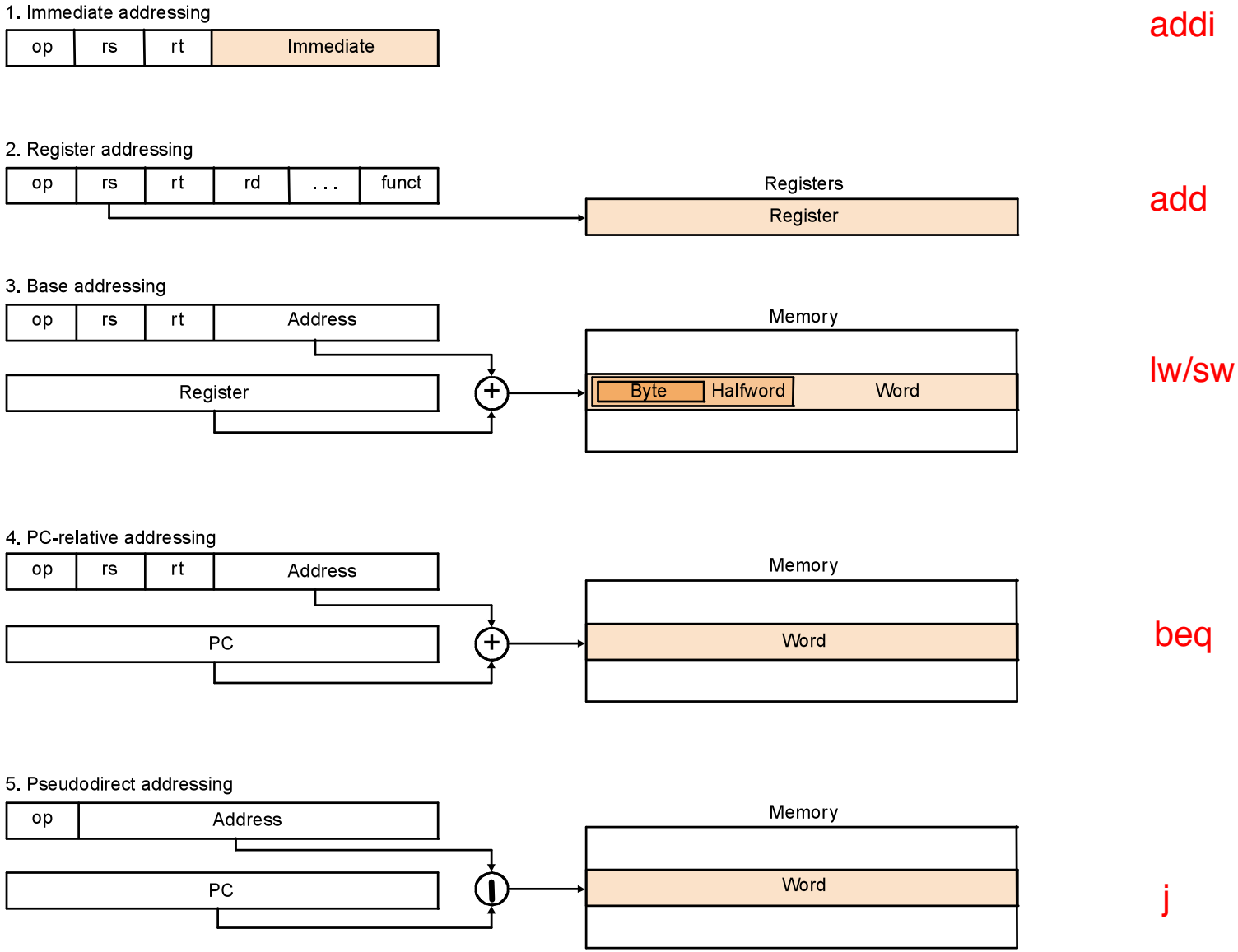
opcode	target address
6b	26b

- Opcode for J-Formats is 2 or 3.
- Key Concepts
  - Keep opcode field identical to R-format and I-format for consistency.
  - Combine all other fields to make room for large target address.
- For now, we can specify 26 bits of the 32-bit bit address.
- Note that, just like with branches, jumps will only jump to word aligned addresses, so last two bits are always 00 (in binary).

J-Format Instructions

- Now, 28 bits out of the 32-bit address have been specified.
- Where do we get the other 4 bits?
  - By definition, take the 4 highest order bits from the PC.
  - Technically, this means that we cannot jump to anywhere in memory, but it's adequate 99.9999…% of the time, since programs aren't that long
    - only if straddle a 256 MB boundary
  - If we absolutely need to specify a 32-bit address, we can always put it in a register and use the jr instruction (which is an R-Format instruction).
- Summary:
  - New PC = { PC[31..28], target address, 00 }
- Understand where each part came from!
- Note: { , , } means concatenation
  { 4 bits , 26 bits , 2 bits } = 32 bit address
  - { 1010, 11111111111111111111111111, 00 } = 10101111111111111111111111111100
- Note: Book uses the symbol '||'

MIPS Addressing Modes



Summary of MIPS Instruction Formats

- MIPS Machine Language Instruction:
  32 bits representing a single instruction

R

opcode	rs	rt	rd	shamt	funct
6b	5b	5b	5b	5b	6b

I

opcode	rs	rt	immediate
6b	5b	5b	16b

J

opcode	target address
6b	26b

- Branches use PC-relative addressing
- Jumps use absolute addressing

Disassembling Machine Instructions

- How do we convert 1's and 0's to C code?
  - Machine language ⇒ C?
- For each 32-bit word:
  - Look at opcode: 0 means R-Format, 2 or 3 mean J-Format, otherwise I-Format.
  - Use instruction type to determine which fields exist.
  - Write out MIPS assembly code, converting each field to name, register number/name, or decimal/hex number.
  - Logically convert this MIPS code into valid C code. Always possible? Unique?
- Example: Here are six machine language instructions in hexadecimal:

  0x00001025
  0x0005402A
  0x11000003
  0x00441020
  0x20A5FFFF
  0x08100001
- Let the first instruction be at address 4,194,304ten (0x00400000).

Example (cont'd)

- The six machine language instructions in binary:

00000000000000000001000000100101	0x00001025
00000000000001010100000000101010	0x0005402A
00010001000000000000000000000011	0x11000003
00000000010001000001000000100000	0x00441020
00100000101001011111111111111111	0x20A5FFFF
00001000000100000000000000000001	0x08100001

- Next step: identify opcode and format

R	opcode	rs	rt	rd	shamt	funct
	0	5b	5b	5b	5b	6b
I	opcode	rs	rt	immediate		
	1, 4-31	5b	5b	16b		
J	opcode	target address				
	2 or 3	26b				

00000000000000000001000000100101	0x00001025
00000000000001010100000000101010	0x0005402A
00010001000000000000000000000011	0x11000003
00000000010001000001000000100000	0x00441020
00100000101001011111111111111111	0x20A5FFFF
00001000000100000000000000000001	0x08100001

Example (cont'd)

- Look at opcode: 0 means R-Format, 2 or 3 mean J-Format, otherwise I-Format. Next step: separation of fields
- Fields separated based on format/opcode:

Format						
R	0	0	0	2	0	37
R	0	0	5	8	0	42
I	4	8	0	+3		
R	0	2	4	2	0	32
I	8	5	5	-1		
J	2	1,048,577				

- Next step: translate ("disassemble") to MIPS assembly instructions

Example (cont'd)

- MIPS Assembly (Part 1):
  - Address: Assembly instructions:

```
0x00400000    or   $2, $0, $0
0x00400004    slt  $8, $0, $5
0x00400008    beq  $8, $0, 3
0x0040000c    add  $2, $2, $4
0x00400010    addi $5, $5, -1
0x00400014    j    0x100001
```

0	0	0	2	0	37
0	0	5	8	0	42
4	8	0	+3		
0	2	4	2	0	32
8	5	5	-1		
2	1,048,577				

- Better solution: translate to more meaningful MIPS instructions (fix the branch/jump and add labels, registers)
- MIPS Assembly (Part 2):

```
          or   $v0, $0, $0
Loop:     slt  $t0, $0, $a1
          beq  $t0, $0, Exit
          add  $v0, $v0, $a0
          addi $a1, $a1, -1
          j    Loop
Exit:
```

- Note: PC for j is:
  - { PC[31..28], target address, 00 } = 0000 01 0000 0000 0000 0000 0001 00
- Next step: translate to C code.

Example (cont'd)

Before

```
0x00001025
0x0005402A
0x11000003
0x00441020
0x20A5FFFF
0x08100001
```

```
        or   $v0,$0,$0
Loop:   slt  $t0,$0,$a1
        beq  $t0,$0,Exit
        add  $v0,$v0,$a0
        addi $a1,$a1,-1
        j    Loop
Exit:
```

- After C code (Mapping below)
  - $v0: product
  - $a0: multiplicand
  - $a1: multiplier
- product = 0;
  while (multiplier > 0) {
  product += multiplicand;
  multiplier -= 1;
  }