MIPS ASSEMBLY PROGRAMMING LANGUAGE VIII - PART II

Ayman Hajja, PhD

BRANCHING INSTRUCTIONS

- beq and bne
  - Need to specify a target address if branch taken
  - Also specify two registers to compare
  - Use I-Format

opcode	rs	rt	immediate

  - opcode specifies beq or bne
  - rs and rt specify registers

BRANCHING INSTRUCTIONS

- Branches typically used in loops (if/else, while, for)
  - Loops are generally small
  - Function calls and unconditional jumps handled with jump instruction (J-Format)
- Recall; Instructions stored in a localized area of memory (Code/Text)
  - Largest branch distance limited by size of code
  - Address of current instruction stored in Program Counter

PC-RELATIVE ADDRESSING

- PC-Relative Addressing; Use the immediate fields as a two's compliment offset to PC
  - Branches generally change the PC by a small amount
  - Can specify 2^{16} - 1 addresses from the PC

PC-RELATIVE ADDRESSING

- PC-Relative Addressing; Use the immediate fields as a two's compliment offset to PC
  - Branches generally change the PC by a small amount
  - Can specify 2^{16} - 1 addresses from the PC

Can we do better?

BRANCHING REACH

- Recall; MIPS uses 32-bit addresses
  - Memory is byte-addressable
- Instructions are word-aligned
  - Address is always multiple of 4 (in bytes), meaning it ends with 0b00 in binary
  - Number of bytes to add to the PC will always be multiple of 4
- Immediate specifies words instead of bytes
  - Can now reach up to 2^{16} instructions (2^{18} bytes around PC)

BRANCH CALCULATION

- If we don't take the branch:
  - Next instruction will be PC + 4
- If we do take the branch;
  - Next instruction will be (PC + 4) + (immediate*4)
- Observations;
  - Immediate is number of instructions to jump (remember, specifies word) either forward (positive) or backwards (negative)

BRANCH EXAMPLE

- MIPS Code;

```
        Loop:
        1000  beq $9, $zero, End
        1004  add  $8, $8, $10
        1008  addi $9, $9, -1
        1012  j Loop
        End:
```

- I-Format fields;
  - opcode = 4 (look up on Green Sheet)
  - rs = 9 (first operand)
  - rt = 0 (second operand)
  - What's the immediate value (End)?

BRANCH EXAMPLE

- MIPS Code;

```
Loop:
1000  beq $9, $zero, End
1004  add  $8, $8, $10
1008  addi $9, $9, -1
1012  j Loop
End:
```

4	9	0	3

- I-Format fields;
  - opcode = 4 (look up on Green Sheet)
  - rs = 9 (first operand)
  - rt = 0 (second operand)
  - What's the immediate value (End)? 3

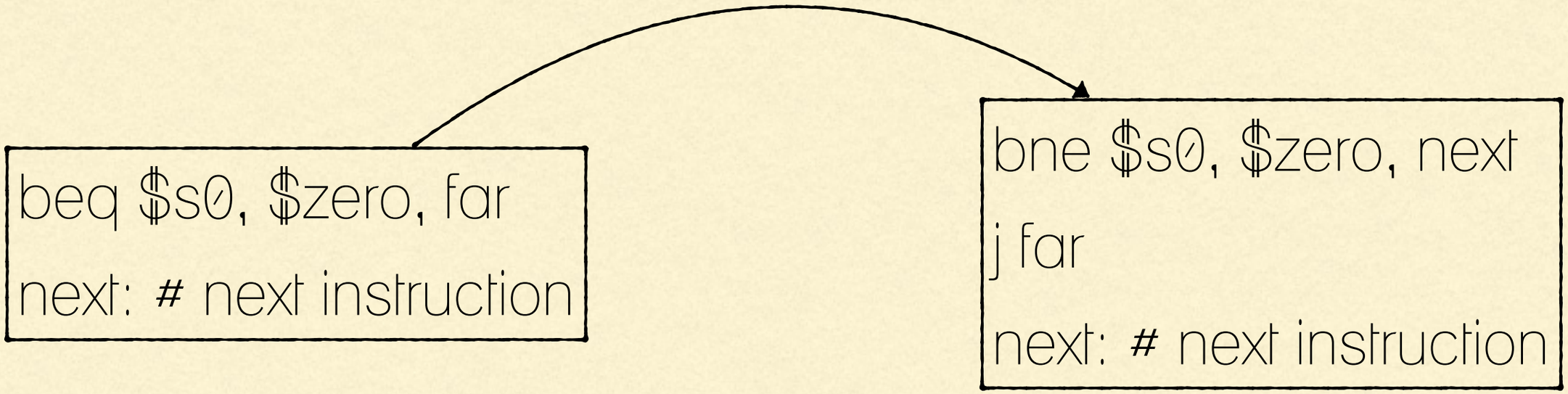
BRANCH EXAMPLE

- What do we do if destination is more than 2^{15} instructions away from branch?

```
beq $s0, $zero, far
next: # next instruction
```

BRANCH EXAMPLE

- What do we do if destination is more than 2^{15} instructions away from branch?



J-FORMAT INSTRUCTIONS

- For branches, we assume that we don't want to branch too far, so we can specify a relative address from PC
- For general jumps (j and jal), we may jump to anywhere in memory;
  - Ideally, we'd like to 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

J-FORMAT INSTRUCTIONS

- Define two "fields" of these bit widths:

6 bits	26 bits

- As usual, each field has a name:

opcode	target address

J-FORMAT INSTRUCTIONS

- We can specify 2^{26} addresses
  - Still going to word-aligned instructions, so add `0b00` as last two bits (multiply by 4)
  - This brings us to 28 bits of a 32-bit address
- If necessary, use jr (R-Format) instead

SUMMARY OF MIPS INSTRUCTIONS FORMAT

- I-Format: instructions with immediates, lw/sw (offset is immediate), and beq/bne
  - Not the shift instruction
  - Branches use PC-relative addressing

- J-Format: j and jal (but not jr) — Jumps use absolute addressing

- R-Format: all other instructions

INTEGER MULTIPLICATION

- Syntax of Multiplication:
  - mult register1, register2
- 32-bit value x 32-bit value = 64-bit value
- Multiplies 32-bit values in those registers and puts 64-bit product in special result registers:
  - Puts product upper half in hi, lower half in lo
  - Recall that 'hi' and 'lo' are two registers separate from the 32 general purpose registers
- Use 'mfhi' and 'mflo' to move from hi, lo to another register

INTEGER DIVISION

- Syntax for Division:
  - div register1, register2
- Divides 32-bit register1 by 32-bit register2:
  - Puts remainder of division in 'hi', quotient in 'lo'