


COMPUTER ORGANIZATION


CODE TRANSLATION II

PROF H ROUMANI
 Dept. of Electrical Engineering and Computer Science, York University
 

PSEUDO INSTRUCTIONS

They are not real, merely conveniences added by the assembler. They're not portable and have no machine language encoding. For Example, `mul`. Others:

```

nop    sll $0, $0, 0
li     addi, ori, lui
abs    slt, beq, sub (with $at)
  
```

Do NOT use pseudo in this course!

2

PSEUDO EXERCISES

```

.text
li     $t0, 5
move   $t0, $t1
blt    $t0, $t1, here
mul    $t5, $t0, $t1
la     $a0, prompt
lw     $t0, big($t0)
  
```

Use \$at if needed for temporary storage

3

FP SUPPORT

- **New Directives in .data**
.float and .double (note the alignment)
- **New Register Set**
\$f0-\$f31 with even-odd pairing
- **New I/O System Calls**
syscall 2, 3, 6, 7
- **Co-Processor 1**
The FP Instruction Subset

4

FP HIGHLIGHTS

- **No FP Immediate**
It won't fit!
- **O/F Handling**
Size doubling not the common case.
- **Internal Transfers**
Bit-Based. Needs representation conversion
- **Flag-Based Branching**
Compare instructions set/reset a flag

5

FP EXERCISE

1. Read a float from the user; multiply it by 4; and then output the product.
2. As #1 but read the float from .data, given its name.
3. As #1 but assume the FP processor is damaged; use the integer CPU.
4. As #1 but for a double.

6

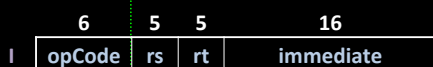
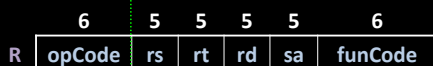
M/L: MACHINE LANGUAGE

- Instruction mnemonics
- Registers
- Numeric, char, or string constants
- Labels
- Variables

Translate all tokens to bits
Exactly 32 bits per instruction

7

M/L: INSTRUCTION FORMATS



Register *rs* = source, *rt* = target, *rd* = destination.

8

M/L: HANDLING LARGE IMMEDIATES

- Branch Labels
 - Drop two LS bits (CPU will add them)
 - PC-Relative Addressing + Branch Reversal
- Jump Labels
 - Drop two LS bits (CPU to return them)
 - Limit .text to 256MB
- Static Vars Addresses (.data)
 - Use lui together with \$gp (compiler)
- Constants
 - Use lui (programmer)

9

M/L: PITFALL

• Branches

spim has a bug in that its PC points at the instruction being executed rather than the one after. Hence, its branches are off by 1.

Example: (correct answer is: ?)

```
loop: add $at, $at, $a0
      div $at, $s0
      mflo $a0
      bne $a0, $0, loop
```

10

M/L: PITFALL

• Jumps

We know how 26 becomes 28 but how does 28 become 32? Answer: Borrow the upper 4 bits from PC.

Example: (correct answer is: ?)

```
12345678      j    done
1234567c      div  $at, $s0
12345680      mflo $a0
12345684 done: jr   $ra
```

11

M/L: PITFALL

• Pseudo

The make things very confusing. Do not ever use them!

Example: (correct answer for loop is: ?)

```
loop: add $at, $at, $a0
      abs $a0, $at
      lw  $at, count($0)
      bne $a0, $0, loop
```

12

CONCURRENCY: THE PROBLEM

- Processing Deposits and Withdrawals
Read-Edit-Write
- Crossing a Two-Way Street
Starvation and Deadlock
- The Dining Philosophers
https://en.wikipedia.org/wiki/Dining_philosophers_problem



11

CONCURRENCY: THE SOLUTION

- Atomicity and Critical Region
A set of operations that shouldn't be interrupted. They should be executed as an all-or-nothing transaction.
- Semaphore = Mutex = Lock
Wait for the lock to be free; lock; read and write; release.
- The MIPS approach
Read via *load-link* with no guarantee of atomicity but the subsequent *store-conditional* will only succeed if all is well.

```
ll $t1, 0($s0) # will always work, but
sc $t0, 0($s0) # works (and sets t0=1) only if the pair is atomic else t0=1
```

14

CONCURRENCY: THE SOLUTION

thread-safe (synchronized) balance withdrawal

```
.text
...
...
# $s0 = address of a/c balance
# $s1 = withdrawal amount

try: ll    $t1, 0($s0)    # load link the balance
     sub   $t1, $t1, $s1  # adjust it
     sc    $t1, 0($s0)    # store it back conditionally
     beq   $t1, $0, try   # did it work?
```

15
