CS 315 – Computer Architecture

MIPS Instruction Set Architecture
(ISA)

MIPS Data Types

- Bytes (signed and unsigned)
- “Halfwords” (signed and unsigned)
- Words (signed and unsigned)
- Floats

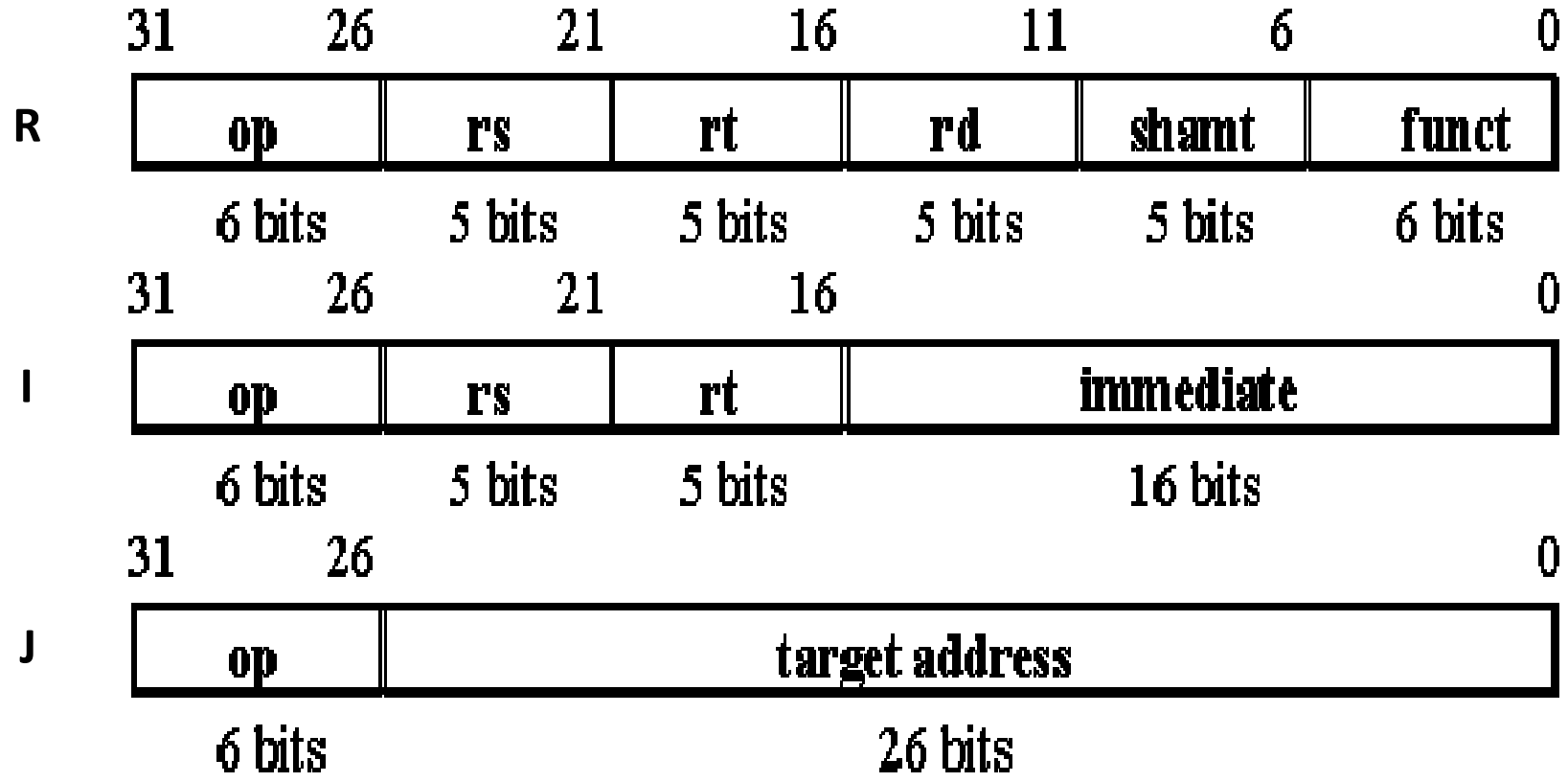
Sign Extension and Arithmetic

- Need to extend sign on numerical values, not necessarily on data or “logical” values
- Shift Right Logical vs. Shift Right Arithmetic.
- Double Shift right – carry between registers.

MIPS Registers (32)

- $zero - $0 – always zero
- $v0-$v1 - $2, 3 – Function results
- $a0 $a3 – 4-7 – Arguments
- $t0-$t7 – temporaries – not saved
- $s0–s8 – Saved by callee in function call
- $t8-t9 – more temporaries
- $gp (28) – global pointer; $sp (29) – stack pointer
- $fp (30) – frame ptr; $ra (31) – return address

MIPS Instruction Format



MIPS Reference Sheet

- See also the MIPS reference sheet for a complete list of MIPS instructions, including those defined for floating point.

- Contained on Polylearn under “Reference Materials”

MIPS Instruction Types

- Register Ops (R format)

add rd, rs, rt - add rs+rt, result to rd

- Immediate (I format)

- addi rt, rs, imm16
- load/store to memory(memory reference):
  lw rt, address (mem [[rs] + offset16])
- branch to a relative location:
  bne rs, rt, offset16 (offset*4 added to pc)

MIPS Instruction Types, cont'd

- Jump Operations (J format)

  j location # anywhere

This provides a 26-bit address, which is then multiplied by 4 to obtain a 28-bit address. Unconditional jump to anywhere in the same bank of 16MBytes.

(Note: Another form of jump is "jr and jalr", which are register format), and allow jump to anywhere within 4GBytes. See also the conditional branches, bne and beq, which are I format.)

Register instructions

- add rd = rs + rt - add (signed)
- addu – add (unsigned)
- sub – subtract (signed)
- subu – subtract (unsigned)
- slt – set if less than rd = (rs < rt) ? 1 : 0
- sltu – set if less than (unsigned)
- jr – jump to address in register (rs)
- jalr – jump and link register (rs)

Register Instructions, cont'd

- and rd, rs, rt # and rs with rt, result to rd
- or rd, rs, rt
- xor rd, rs, rt

Shift Instructions (R format)

Logical and Arithmetic types of Shifts, fixed number of positions:

srl rd, rt, shamt # Shift (shamt) bits right logical

-Shifts register Rt (shamt) locations right, places the result in register Rd. Shifts in 0 into leftmost bit(s).

sra rd, rt, shamt # Shift (shamt) bits right arithmetic (sticky sign)

sra rd, rt, shamt # Shift (shamt) bits right arithmetic (sticky sign)

(Note: There is no sla instruction).

Variable number of shifts:

srlv rd, rt, rs # Shifts register rt >> (value of rs)

sllv "

srav "

Load/Store Instructions (I)

- lw rt = M[offset + $rs], Load Word
  - example: lw $t0, 4($t1)
- lb, lbu – Load byte
- lh, lhu – Load halfword
- sw - Store Word
- lui – Load upper
- sb – Store byte
- sh – Store Half (word)
- sw – Store word

All of these instructions follow the I ("immediate") format.

Additional Immediate Instructions

- addi – add immediate
- addiu – add immediate unsigned
- andi – and immediate
- ori – Or word immediate
- xori – Exclusive or word immediate
- slti – Set less than immediate
- sltiu – Set less than immediate unsigned
- beq, bne – Branch on equal, not equal

MIPS Instruction Types, cont'd

- **Branch with offset**

beq rs, offset (to: [PC] + Offset16<<2)

- **Jump, Jump and Link**

j target26 (to: target26 <<2)

jal target26

- **Jump Register, Jump and link register**

jr rs (pc <= [rs]

jalr rs

Pseudo-Instructions

- Generated by the Assembler which produces the machine code
- Can be 1-1 or 1-N. One line of source code can produce one or more machine instructions.
- Format type 'P' on Reference Card
- Example 1: move rd, rs ... becomes:

    or rd, rs, $zero (one instruction)

Example 2: bge $t0, $s0, Label ... becomes

```
slt $at, $t0, $s0
beq $at, $zero, Label
```

Pseudo-Instruction List

- move
- li
- la
- neg
- not
- ror
- rol
- lw
- blt
- ble
- bgt
- bge

... and more possible

e.g., sd (store double)

Pseudo Instruction ‘move’

- move $t1, $t2 becomes “bare metal”:

```
addu $t1, $zero, $t2
```

Note: it could be lots of other possibilities:

```
or   $t1, $zero, $t2   -or-
xor  $t1, $zero, $t2   -or-
xor  $t1, $t2, $zero ... etc.
```

Pseudo-Instruction ‘li’

- Loads an immediate value into a register. If the value is ‘small’ it can be done with a single ‘bare metal’ instruction

  li $t0, 0x22 becomes:

  ```
  ori $t0, $t0, 0x22
  ```

- But it also handles the case of ‘big’ immediate values, with two instructions:

  li $t0, 0x20000033 becomes:

  ```
  lui $at, 0x2000    # get high 16 bits
  ori  $t0, $at, 0x33   # or in the low 16 bits
  ```

la Instruction

- la $t0, main (0x400024) becomes:

```
lui   $at, 0x40    # hi 16 bits of address
ori $t0, $at, 0x24   # low 16 bits
```

neg (negate) pseudo-instruction

- neg $t0, $t1 becomes:

  sub $t0, $zero, $t1

not pseudo-instruction

- This inverts all the bits in a register.
- not $t0, $t1 becomes:

  nor $t0, $t1, $zero

ror (Rotate Right) pseudo-instruction

- ror $t0, $t0, 4 becomes:

```
sll $at, $t0, 28
srl $t0, $t0, 4
or $t0, $t0, $at
```

rol (Rotate Left) pseudo-instruction

- rol $t0, $t0, 3 becomes:

```
srl $at, $t0, 29
sll  $t0, $t0, 3
or $t0, $t0, $at
```

Note: uses the $at ($1) register as a temporary

lw (load word)

- lw $t0, data2, where the value of symbol "data2" is 0x10010004 becomes:

```
lui  $at, 0x1001   # high 16 bits
lw  $t0, 4($at)    # load value
```

- Note offset field of "bare metal" instruction limited to -32768 .. +32767

lw $t0, 0x201000($t4)

- offset > 32767 !!!
- Pseudo-instruction allows it to become:

```
lui  $at, 0x20
addu $at, $at, $t4
lw $t0, 0x1040($at)
```

blt – Branch if Less Than

- blt $t0, $t1, label becomes:

```
slt $at, $t0, $t1
bne  $at, $zero, label
```

ble - Branch if Less than or Equal

- ble $t0, $t1, label becomes:

```
slt  $at, $t1, $t0
beq $at, $zero, label
```

bgt – Branch if Greater Than

- bgt $t0, $t1, label becomes

  slt $at, $t1, $t0

  bne $at, $zero, label

bge – Branch if Greater or Equal

- bge $t0, $t1, label becomes:

```
slt $at, $t0, $t1
beq $at, $zero, label
```

Pseudo-Instructions vs. Macros

- Pseudo-Instructions are pre-defined in the assembler
- Macros are user-defined and allow extensions of the language at the time of assembly.

Conditional Operations (note: not found on MIPS ISA)

- Some machines use “condition codes”

  Z N C V = Zero, Negative, Carry, oVerflow. (NOT MIPS!)

- Some machines test/compare and branch

  E.g., beq $r1, $r2, L1 # if R1==R2, goto L1

- Some machines (ARM) make every instruction conditional!

MIPS Exceptions

- traps – (syscalls) – invoke kernel via a trap vector. (synchronous exceptions)
- error conditions (e.g., overflow on signed addition) (synchronous exception)
- memory management errors (undefined page exception)
- illegal instructions
- hardware errors
- I/O interrupts (asynchronous exceptions)

General Addressing

- Not just looking at MIPS here – there are many ways of implementing access to memory.
- Objective is to generate an *effective address*.

General Addressing Modes, cont'd.

- Direct Memory – limited access
- Direct Memory (paged) – relative to PC
- PC + Offset – requires n bits for Offset *
- Indirect (memory loc)
- Register Memory pointer auto-increment
- Register Memory point auto-decrement
- Stack

General Addressing Modes

- Register ([Rn]) – takes only 2^n bits of instr. *
- Immediate Data - value (n bits of instr.) *
- Immediate Data + Reg. (add Reg to imm.) *
- Memory M[Reg + Offset] *
- Base + Index + Offset (great for indexing)

* supported by MIPS