32-bit Constants

- Most constants are small
  - 16-bit immediate is sufficient
- For the occasional 32-bit constant
  - lui rt, constant 16 bit
  - Copies 16-bit constant to left 16 bits of rt
  - Clears right 16 bits of rt to 0

32 bit

lui $s0, 61	0000 0000 0111 1101 0000 0000 0000 0000
ori $s0, $s0, 2304	0000 0000 0111 1101 0000 1001 0000 0000

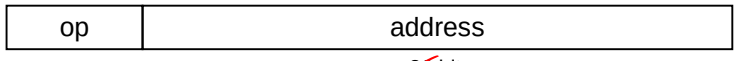
Branch Addressing

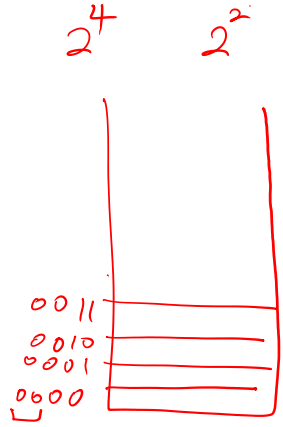
- Branch instructions specify
  - Opcode, two registers, target address
- Most branch targets are near branch
  - forward / backward
- PC-relative addressing
  - Target address = PC + offset x 4
  - PC already incremented by 4 by this time

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits 18

Jump Addressing

- Jump (j and jal) targets could be anywhere in text segment
  - Encode full address in instruction
- (Pseudo) Direct jump addressing
- Target address = $PC_{31...28}$: (address × 4)
  - MIPS program size limitation: 256 MB





CSULB

Target Addressing Example

- Loop code from earlier example
  - Assume Loop at location 80000

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

80000	0	0	19	9	4	0	
80004	0	9	22	9	0	32	
80008	35	9	8	0			
80012	5	8	21				
80016	8	19	19	1			
80020	2						
80024							

PC

CSULB

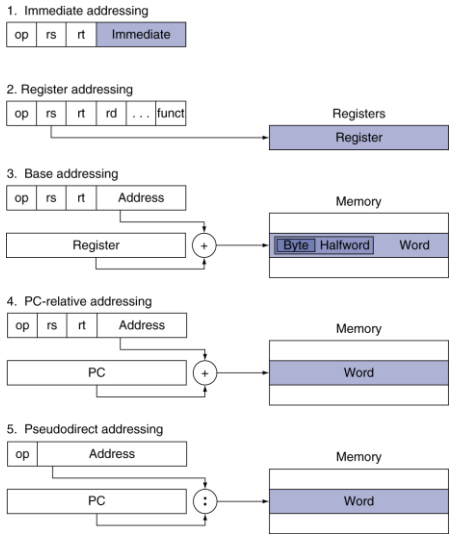
Branching Far Away

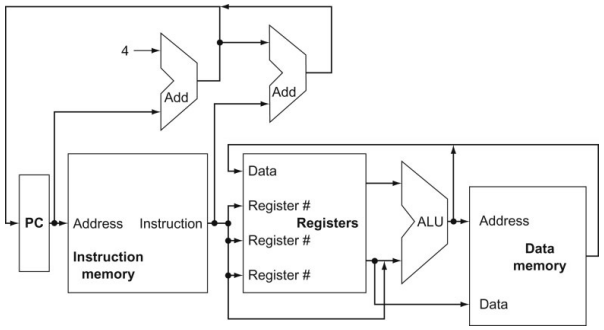
- If branch target is too far to encode with 16-bit offset, assembler rewrites the code
- Example

(Assume L1 is out of range with $\pm 2^{15}$ offset)

```
      beq $s0,$s1, L1
            ↓
      bne $s0,$s1, L2
      j L1
L2:   …
```


Addressing Mode Summary





Synchronization

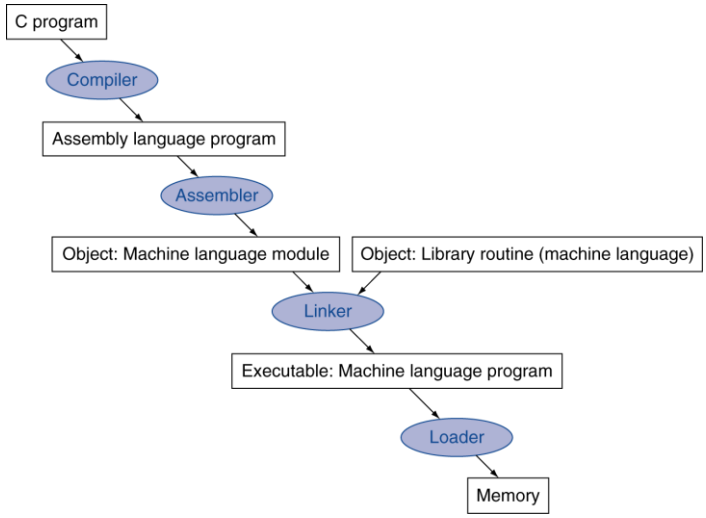
- Two processors sharing an area of memory
  - P1 writes, then P2 reads
  - Data race if P1 and P2 don't synchronize
    - Result depends of order of accesses
- Hardware support required
  - Atomic read/write memory operation
  - No other access to the location allowed between the read and write
  - By providing an atomic pair of instructions

Synchronization in MIPS

- Load linked: `ll rt, offset(rs)`
- Store conditional: `sc rt, offset(rs)`
  - Succeeds if location not changed since the `ll`
    - Returns 1 in rt
  - Fails if location is changed
    - Returns 0 in rt
- Example: atomic swap (to test/set lock variable)

```
try: add $t0,$zero,$s4 ;copy exchange value
     ll  $t1,0($s1)    ;load linked
     sc  $t0,0($s1)    ;store conditional
    … something's going on by other processor…
     beq $t0,$zero,try ;branch store fails
     add $s4,$zero,$t1 ;put load value in $s4
```

Translation and Startup



Assembler Pseudoinstructions

- Most assembler instructions represent machine instructions one-to-one
- Pseudoinstructions: figments of the assembler's imagination

```
move $t0, $t1      → add $t0, $zero, $t1

blt $t0, $t1, L    → slt $at, $t0, $t1
                     bne $at, $zero, L
```

  - $at (register 1): assembler temporary

Object Modules

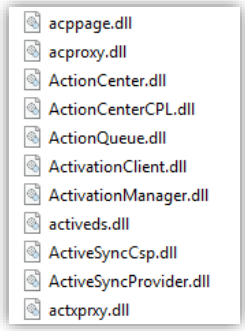
- Assembler (or compiler) translates program into machine instructions
- Produces an executable images by
  - Merges segments (object modules)
  - Resolve labels (determine their addresses)
  - Patch location-dependent and external refs

Loading a Program

- Load from image file on disk into memory
  - Read header to determine segment sizes
  - Create virtual address space
  - Copy text and initialized data into memory
    - Or set page table entries so they can be faulted in
  - Set up arguments on stack
  - Initialize registers (including $sp, $fp, $gp)
  - Jump to startup routine
    - Copies arguments to $a0, … and calls main
    - When main returns, do exit syscall

Dynamic Linking

- Only link/load library procedure when it is called
  - Requires procedure code to be relocatable
  - Avoids image bloat caused by static linking of all (transitively) referenced libraries
  - Automatically picks up new library versions



9/19