CS61C
Other Instruction Sets: HP-PA and Intel x86

Lecture 22

April 16, 1999

Dave Patterson (http.cs.berkeley.edu/~patterson)

www-inst.eecs.berkeley.edu/~cs61c/schedule.html

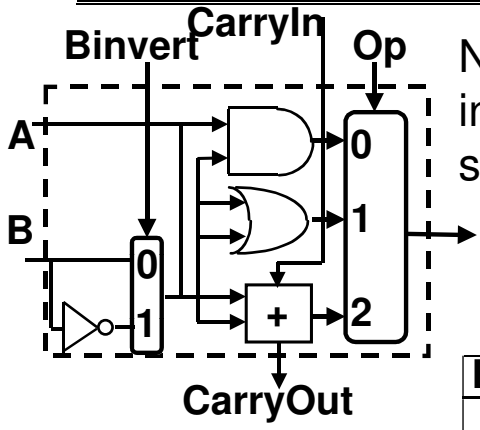
Outline

- Review Datapath, ALU
- HP-PA vs. MIPS
- Example: HP-PA vs. MIPS
- Administrivia, "Computers in the News"
- 80x86 History
- 80x86 instructions vs. MIPS
- Example: 80x86
- Conclusion

Review 1/2

- Subtract included to ALU by adding one's complement of B
- Multiple by shift and add
- Divide by shift and subtract, then restore by add if didn't fit
- Can Multiply, Divide simply by adding 64-bit shift register to ALU
- MIPS allows multiply, divide in parallel with ALU operations

Review 2/2: 1-bit ALU with Subtract Support



Note: And, Or, Add occur in parallel, with multiplexor selecting the desired result

Definition

Binvert	Op	C
0	0	A and $\overline{B}$
1	0	A and $\overline{B}$
0	1	A or B
1	1	A or $\overline{B}$
0	1	A + $\overline{B}$ + CarryIn
1	1	A + $\overline{B}$ + CarryIn

MIPS vs. HP-PA

MIPS	HP-PA
° Address: 32-bit	° 32-bit
° Page size: 4KB	° 4KB
° Data aligned	° Data aligned
° Destination reg: Left	° Right
• `add $rd,$rs1,$rs2`	• `addo %rs1,%rs2,%rd`
° Regs: $0, $1, ..., $31	° %r0, %r1, ..., %r31
° Reg = 0: $0	° %r0
° Return address: $31	° %r2

Instructions:MIPS vs. HP-PA

° `addu, addiu`	° `addl, addi`
° `subu`	° `subl, subi`
° `and,or, xor`	° `and, or, xor`
° `lw`	° `ldw (load word)`
° `sw`	° `stw (store word)`
° `mov`	° `copy`
° `li`	° `li`
° `lui`	° `ldil(load imm left)`

Branch: MIPS vs. HP-PA

° `beq`	° `cmpb,=` compare & branch
° `bne`	° `cmpb,<>` less or greater (!=)
° `slt; beq`	° `cmpb,<`
° `slt; bne`	° `cmpb,>=`
° `jal`	° `bl L0,$r2` branch & link into r2
° `jr $31`	° `bv 0(%r2)` branch via r2

Unique Instructions

° `ldo` (load offset)

- Calculate address like a load, but load address into register, not data
- Load 32-bit constant:

```
ldil left_const,%rx
ldo  right_const(%rx),%rx
```

HP-PA data addressing

- ldw: base reg + offset (like MIPS)
  - ldw 4608(0,%r19),%r25 # r19+4608
- ldwx base reg + index (unlike MIPS)
  - ldwx %r20(0,%r19),%r25 # r19+r20
- scaled reg + offset (unlike MIPS)
  - ldw,s 12(0,%r19),%r25 # (r19<<2)+12
  - Purpose: to turn index into byte address, shift left register by 0, 1, 2, 3 for byte, halfword, word, doubleword data transfer
- scaled reg + index (ldwx,s)

HP-PA data addressing (Cont'd)

- Update register with calculated address as part of instruction ("autoupdate")
  - ldw 4608(1,%r19),%r25
    # r25=Mem[r19+4608]; r19=r19+4608
  - ldwx %r20(1,%r19),%r25
    # r25=Mem[r19+d20]; r19=r19+r20
  - ldw,s 8(1,%r2),%r4
    # r4=Mem[(r2<<2)+8]; r2=(r2<<2)+8
  - ldwx,s r3(1,%r2),%r4
    # r4=Mem[(r2<<2)+r3];r2=(r2<<2)+r3
- Purpose: fewer instructions ⇒ performance

While in C/Assembly: MIPS

C

```
while (save[i]==k)
   i = i + j;
```

(i,j,k: $s3,$s4,$s5)

MIPS

```
      addi $s6,$sp,504# save[]
Loop:sll $t0, 2,$s3   #$t0 = 4*i
     add $t0,$t0,$s6  #$t0=Addr
     lw  $t1,0($t0)   #$t1=save[i]
     bne $t1,$s5,Exit#goto Exit
                     #if save[i]!=k
     add $s3,$s3,$s4  # i = i + j
     j   Loop         # goto Loop
Exit:
```


While in C/Assembly: HP-PA

C

```
while (save[i]==k)
   i = i + j;
```

(i,j,k: %r3,%r4,%r5)

HPPA

```
      ldo -504(%r30),%r7 # save[]
Loop:ldwx,s %r3(0,%r7),%r6 #save[i]
     comb,<> %r5,%r6,Exit
     addl %r3,%r4,%r3 # i = i + j
     b    Loop        # goto Loop
Exit:
```

Note: ldwx,s replaces sll, add, lw in loop

HP-PA Unique Instructions: Shift Left

- `zdep %rs,pos,len,%rd`
  - deposit right-adjusted field of width `len` to bit `pos`, and zero rest of register
  - MIPS: `sll $rd,2,$rs`
    `= zdep %rs,31-2,32-2,%rd`
    `= zdep %rs,29,30,%rd`
- `zvdep %rs, len,%rd`
  - deposit right-adjusted of width `len` to bit specified in reg `sar`, and zero rest
  - MIPS: `sllv $rd,%rv,$rs`
    `= mtsar %rv; zvdep %rs,32,%rd`
- `extr`, `vextr` is opposite; extracts

HP-PA Unique Instructions

- `extru %rs,pos,len,%rd`
  - extract field of width `len` at bit `pos` & place right-adjusted into register, and zero rest (`extrs` sign extends; `vextrs` uses `sar`)
  - MIPS: `srl $rd,2,$rs`
    `= extru %rs,31-2,32-2,%rd`
    `= extru %rs,29,30,%rd`
- Shift left 1, 2, or 3 bits and add
  - Purpose: provide a primitive operations for multiply so that can multiply by constants more efficiently
  - MIPS: `sll $rx,2,$rs; add $rd,$rx,$rt`
    `= sh2addl %rs,%rt,%rd`

HP-PA Floating Point

- `fldws` - load word into FP reg
- `fsub,sgl` - SP FP subtract
- `fdiv,sgl` - SP FP divide
- `fcnvxf,sgl,sgl` - convert int to SP FP
- `fstws`- store word from FP reg
- 58 Single Precision floating point registers, called
  `%fr4L`, `%fr4R`, `%fr5L`, `%fr5R`, ...,
  `%fr30L`, `%fr30R`, `%fr31L`, `%fr31R`

Administrivia

- Project 6: MIPS sprintf; Due Wed April 28
- Next Readings: 2.1 to 2.5
- 9th homework: Due Today (Ex. 7.35, 4.24)
- 10th homework: Due Wednesday 4/21 7PM
  - Exercises 4.43, 3.17 (assume each instruction takes 1 clock cycle, performance = no. instructions executed * clock cycle time, ignore CPI comment)

Administrivia: Rest of 61C

W 4/21 Performance; Reading sections 2.1-2.5
F 4/23 Review: Procedures, Variable Args
(Due: x86/HP ISA lab, homework 10)

W 4/28 Processor Pipelining; section 6.1
F 4/30 Review: Caches/TLB/VM; section 7.5
(Due: Project 6-sprintf in MIPS, homework 11)

M 5/3 Deadline to correct your grade record

W 5/5 Review: Interrupts/Polling
F 5/7 61C Summary / Your Cal heritage
(Due: Final 61C Survey in lab)

Sun 5/9 Final Review starting 2PM (1 Pimintel)

W 5/12 Final (5PM 1 Pimintel)

- Need Alternative Final? Contact mds@cory

"Computers in the News"

- "Computer Age Gains Respect of Economists", N.Y. Times, April 14, 1999
  - 1990: "You can see the Computer Age everywhere but in the productivity statistics," Robert Solow, a MIT Nobel prizewinner
  - Productivity growth has picked up...2% 1995-98 v. 1% for 1974-95.... apparently having to do with the increased speed & efficiency that the Internet and other pervasive information technology advances for mundane businesses operations
  - Greenspan: 1999 economy enjoying "higher, technology-driven productivity growth."
  - Solow:"My beliefs are shifting on this subject"

Intel History

- Intel evolved since 1978
  - 8086: 16-bit, all internal registers 16 bits wide; no general purpose registers; '78
  - 8087: + 60 Fl. Pt. instructions, adds 80-bit-wide stack, but no registers; '80
  - 80286: adds elaborate protection model; '82
  - 80386: 32-bit; converts 8 16-bit registers into 8 32-bit general purpose registers; new addressing modes; adds paging; '85
  - 80486, Pentium, Pentium II: + 4 instructions
  - MMX: + 57 instructions for multimedia; '97
  - Pentium III: +70 instructions for multimedia; '99

MIPS vs. 80386

MIPS	80386
Address: 32-bit	32-bit
Page size: 4KB	4KB
Data aligned	Data unaligned
Destination reg: Left	Right
`add $rd,$rs1,$rs2`	`add %rs1,%rs2,%rd`
Regs: $0, $1, ..., $31	%r0, %r1, ..., %r7
Reg = 0: $0	(n.a.)
Return address: $31	(n.a.)

MIPS, HP-PA vs. Intel 80x86

- MIPS, HP-PA: "Three-address architecture"
  - Arithmetic-logic specify all 3 operands

```
add $s0,$s1,$s2 # s0=s1+s2
```

  - Benefit: fewer instructions ⇒ performance
- x86: "Two-address architecture"
  - Only 2 operands, so the destination is also one of the sources

```
add $s1,$s0 # s0=s0+s1
```

  - Benefit: smaller instructions ⇒ smaller code

MIPS, HP-PA vs. Intel 80x86

- MIPS, HP-PA: "load-store architecture"
  - Only Load/Store access memory; rest operations register-register; e.g.,

```
lw $t0, 12($gp)
add $s0,$s0,$t0 # s0=s0+Mem[12+gp]
```

  - Benefit: simpler hardware ⇒ performance
- x86: "register-memory architecture"
  - All operations can have an operand in memory; other operand is a register; e.g.,

```
add 12(%gp),%s0 # s0=s0+Mem[12+gp]
```

  - Benefit: fewer instructions ⇒ smaller code

MIPS, HP-PA vs. Intel 80x86

- MIPS, HP-PA: "fixed-length instructions"
  - All instructions same size, e.g., 4 bytes
  - simple hardware ⇒ performance
  - branches can be multiples of 4 bytes
- x86: "variable -length instructions"
  - Instructions are multiple of bytes: 1 to 17;

  ⇒ small code size (30% smaller?)
  - More Recent Performance Benefit: better instruction cache hit rates
  - Instructions can include 8- or 32-bit immediates

Unusual features of 80x86

- 8 32-bit Registers have names; 16-bit 8086 names with "e" prefix:
  - `eax`, `ecx`, `edx`, `ebx`, `esp`, `ebp`, `esi`, `edi`
- PC is called `eip` (instruction pointer)
- `leal` (load effective address)
  - Like HP-PA `ldo`
  - Calculate address like a load, but load address into register, not data
  - Load 32-bit address:

```
leal -4000000(%ebp),%esi
 # esi = ebp - 4000000
```

Instructions:MIPS vs. 80x86

° addu, addiu	° addl
° subu	° subl
° and,or, xor	° andl, orl, xorl
° sll, srl, sra	° sall, shrl, sarl
° lw	° movl mem, reg
° sw	° movl reg, mem
° mov	° movl reg, reg
° li	° movl imm, reg
° lui	° n.a.

80386 addressing (ALU instructions too)

- base reg + offset (like MIPS)
  - `movl -8000044(%ebp), %eax`
- base reg + index reg (like HP-PA)
  - `movl (%eax,%ebx),%edi`
    `# edi = Mem[ebx + eax]`
- scaled reg + index (like HP-PA)
  - `movl(%eax,%edx,4),%ebx`
    `# ebx = Mem[edx*4 + eax]`
- scaled reg + index + offset
  - `movl 12(%eax,%edx,4),%ebx`
    `# ebx = Mem[edx*4 + eax + 12]`

Branch in 80x86

- Rather than compare registers, x86 uses special 1-bit registers called "condition codes" that are set as a side-effect of ALU operations
  - S - Sign Bit
  - Z - Zero (result is all 0)
  - C - Carry Out
  - P - Parity: set to 1 if even number of ones in rightmost 8 bits of operation
- Conditional Branch instructions then use condition flags for all comparisons: <, <=, >, >=, ==, !=

Branch: MIPS vs. 80x86

° beq	° (cmpl;) je if previous operation set condition code, then `cmpl` unnecessary
° bne	° (cmpl;) jne
° slt; beq	° (cmpl;) jlt
° slt; bne	° (cmpl;) jge
° jal	° call
° jr $31	° ret

While in C/Assembly: HP-PA

C

```
while (save[i]==k)
   i = i + j;
```

(i,j,k: %edx,%esi,%ebx)

x86

```
      leal -400(%ebp),%eax
.Loop: cmpl %ebx,(%eax,%edx,4)
      jne .Exit
      addl %esi,%edx
      j .Loop
.Exit:
```

Note: cmpl replaces sll, add, lw in loop

Unusual features of 80x86

- Memory Stack is part of instruction set
  - `call` places return address onto stack, increments `esp` (Mem[`esp`]=`eip`+6; `esp`+=4)
  - `push` places value onto stack, increments `esp`
- `pop` gets value from stack, decrements `esp`
- `incl, decl` (increment, decrement)

```
incl %edx # edx = edx + 1
```

  - Benefit: smaller instructions ⇒ smaller code

Unusual features of 80x86

- `cl` is the old count register, & can be used to repeat an instruction; it is 8 rightmost bits of `ecx`
- Used by shift to get a variable shift; uses `cl` to indicate variable shift

```
movl (%esi),%ecx # exc = M[esi]
sall %cl,%eax,%ebx # ebx << exc
```

- Positive constants start with $; regs with %
  - `cmpl $999999,%edx`

Unusual features of 80x86: Floating Pt.

- Floating point uses a separate stack; load, push operands, perform operation, pop result

```
fildl (%esp)
   # fpstack = M[esp],
   # convert integer to FP
flds -8000048(%ebp)
   # push M[ebp-8000048]
fsubp %st,%st(1)
   # subtract top 2 elements
fstps -8000048(%ebp)
   # M[ebp-8000048] = difference
```

MIPS, HP-PA vs. Intel 80x86

- MIPS, HP-PA: "fixed-length operatons"
  - All operations on same data size: 4 bytes; whole register changes
  - Goal: simple hardware and high performance
- x86: "variable -length operations"
  - Operations are multiple of bytes: 1, 2, 4
  - Only part of register changes is < 4 bytes
  - Condition codes are set based on width of operation for Carry, Sign, Zero

"And in Conclusion.." 1/1

- Once you've learned one RISC instruction set, easy to pick up the rest
  - ARM, Compaq/DEC Alpha, Hitatchi SuperH, HP PA, IBM/Motorola PowerPC, Sun SPARC, ...
- HP-PA: more complex than MIPS
- Intel 80x86 is a horse of another color
- RISC emphasis: performance, HW simplicity
- 80x86 emphasis: code size
- Next: Performance