Computer Science 210

Computer Systems 1

2007 Semester 1

Lecture Notes Part 2

Instructions & Addressing Modes

Lecture 7
30 Mar 07

James Goodman

Department of Computer Science

Reminders

- **Mini-assignment 2 is not yet ready. The due date (originally 3April) will be deferred until after the break.**
- **NO CLASS NEXT TUESDAY, 3April!!**
  - **Only class next week: Thursday, 5April**

Optimization: lda, ldah

lda	Reg A	Reg B	<low address>
6 bits	5 bits	5 bits	16 bits

ldah	Reg A	Reg B	<hi address>
6 bits	5 bits	5 bits	16 bits

- **`lda A, <low address>(B)`**
  - sign-extend constant <lowaddress> and add to contents of register B; assign result to register A
- **`ldah A, <hi address>(B)`**
  - multiply constant <hi address> by 65,536 and add to contents of register B; assign result to register A
- *Usually* only requires 2 instructions
  - Still only generates 32-bit addresses
  - Some 32-bit integers cannot be generated this way

Base + Displacement

- Base: a long-term but approximate address
  - Gives location of larger structure
  - Can be dynamically varied
- Displacement
  - Static offset embedded in instruction
  - Cannot be dynamically varied

Load Reg, Disp(Base)

ldq	Dest	Base	Displacement
6 bits	5 bits	5 bits	16 bits

Effective address: (Base) + Displacement

- Base is a 64-bit address
- Displacement is a 16-bit signed constant, sign-extended to 64 bits
- Displacement defines position *relative to* Base

Recommended Readings

- Today's lecture mostly based on Chapters 4 & 5 of Dr. Hutton's notes.

Load Instructions

```
• ldq reg, disp(base)   ! Load quadword
• ldl reg, disp(base)   ! Load sign-extended longword
• ldwu reg, disp(base)  ! Load zero-extended word
• ldbu reg, disp(base)  ! Load zero-extended byte

• stq reg, disp(base)   ! Store quadword
• stl reg, disp(base)   ! Store longword
• stw reg, disp(base)   ! Store word
• stb reg, disp(base)   ! Store byte

• lda reg, disp(base)   ! Assign computed addr to reg
• ldah reg, disp(base)  ! Multiply displacement by
                        ! 65,536 and add to base,
                        ! assign to address
```

Summary: Possible Addressing Modes for Memory Operations

- Direct
  - address contained in instruction
- Indirect
  - Instruction contains address where address is held
- Register indirect
  - Instruction specifies register where address is held
- Register + Register
  - Instruction specifies two registers
  - Contents of registers are added to determine address
- Base + displacement
  - Instruction specifies register and contains displacement
  - Displacement is added to content of register to determine address

Other Possible Addressing Modes

- Immediate operand
  - Instruction contains the value, used as an operand
  - Limited by word size to small constant (8 bits)
  - Example: **`addq $5, 1, $5`**
  - Example: **`lda reg, disp($31)`**

Branch Instruction

- How to specify full add (≤ 53 bits) in a 32-bit instruction?
- Observation: most branches are short
- Branch can use *relative address*: difference from current value of PC.

Example: bne

11 1010	Reg	Target
6 bits	5 bits	21 bits

- Instruction must be aligned: two LSBs must be zero
- Test register specifies 1 of 32 registers for testing
- 21 bits can specify a branch relative to current instruction of PC ± 2^{20} instructions
- New PC = PC + sign-extend(4 * Target)

Long-distance Branches

- Jump instruction
  - Full address specified indirectly through register
  - Unconditional transfer of control

Examples of Operand Specifications

- Register (operate, control, memory)
- Unsigned 8-bit constant (operate instructions)
- Unsigned 6-bit count (shift instructions)
- Base + displacement (memory)
- 21-bit branch offset (control)
- 26-bit constant (PALcode format)

I/O Instructions

- Privileged Architecture Library (PAL)
  - A set of functions of arbitrary complexity invoked by a special call_pal instruction
  - Performs privileged operations such as accessing disk, reading and printing, etc.
- Form: call_pal constant

```
call_pal CALL_PAL_CALLSYS
call_pal CALL_PAL_BPT
```

Simple I/O

- getchar (result in $v0)

```
ldiq $a0, 0x1   // CALLSYS_GETCHAR
call_pal 0x83   // CALL_PAL_CALLSYS
```

- putchar (character in $a1)

```
ldiq $a0, 0x2   // CALLSYS_PUTCHAR
call_pal 0x83   // CALL_PAL_CALLSYS
```

Registers Named

$0	$v0
$1-$8,	$t0-$t9
$9-$14	$s0-$s5
$15	$fp
$16-$21	$a0-$a5
$22-$25	$t8-$t11
$26	$ra
$27	$pv
$28	$at
$29	$gp
$30	$sp
$31	$zero (special)

Register Names

\$t0-\$t11	Temporary registers, used to hold temporary values, when evaluating expressions, etc.
\$s0-\$s5	Saved registers, used to hold the values of local variables in functions.
\$a0-\$a5	Argument registers, used to pass parameters to functions.
\$v0	Value register, used to return the result of a function.
\$ra	Return address register, used to hold the return address of a function.
\$gp	Global pointer register, used to point to the table of constants.
\$sp	Stack pointer register, used to point to the top of the stack used to allocate space for functions.
\$zero	Zero register, that always contains the value zero. Attempting to write to this register has no effect.