Computer Science 210

Computer Systems 1

2007 Semester 1

Lecture Notes

Load/Store Instructions

Lecture 6
29 Mar 07

James Goodman

Department of Computer Science

Recommended Readings

- Today's lecture mostly based on Section 4.2 of Dr. Hutton's notes.

Assignment 2 Delayed

Mini-assignment 2 is not yet ready.

The due date (originally 3April) will be deferred until after the break.

Load & Stores

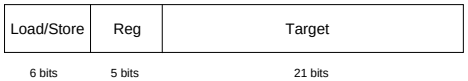
```
ldq Reg, Address  ! Direct
```

What's the problem?

- Address is stored as a constant inside the instruction
  - How to *dynamically* change the address?
- Instructions should be small, fixed size
  - Addresses are large
  - How to store a 51-bit address in a 32-bit instruction?
- Also a problem for branch instructions:

```
bne Reg, Address
```


Load/Store Instructions



- How to specify memory address (***effective address***) with only 21 bits?

Idea: Add another word

- Load/store instructions could be 64 bits
- 21 + 32 = 53 bits

But

- Instructions are not all the same size
- Load address is a constant—can't be changed within program

Idea 2: Address is in Register

- Load instruction specifies a register to supply address: Register Indirect
- Easy to change address without changing instruction

But

- How does the address get into the register?
- How is the address modified?

Idea 3: Construct Effective Address

- lda instruction loads constant into register

```
lda reg, constant
```

- Can adjust address dynamically (using addition)
- 21-bit constant is not big enough
- Assume <constant> is 32 bits
- Break <constant> into two 16-bit pieces:

```
16 most significant bits: <hi address>
16 least-significant bits: <low address>
```


Idea 3: Construct Effective Address

- Use instruction sequence:

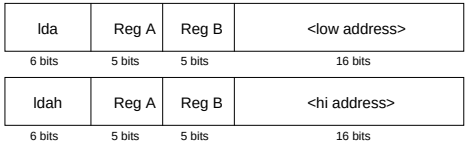
```
lda $8, <hi address>
sll $8, $8, 16
lda $9, <low address>
bis $8, $9, $10     ! Logical OR
ldq $11, ($10)      ! Register indirect
```

- Five instructions!
- Variant: use add instead of OR
  - sign-extend <low address>

```
add $8, $9, $10     ! Instead of Logical OR
```

  - Doesn't quite work if <low address> is negative!

Optimization: lda, ldah



- **`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

Locality of Reference

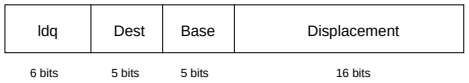
Observation: memory references are not random

- Accesses tend to be clustered
  - in time (temporal locality)
  - in space (spatial locality)
- Accesses are to objects
  - structures
  - arrays
- Can dynamic compute address arithmetically
- Can statically predict offset within known structure

Idea 4: Use Combination

- 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)



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

Special Case of Base + Displacement

- Register Direct
  - Zero displacement
  - Register specifies address