Computer Science 210

Computer Systems 1

2007 Semester 1

Lecture Notes

Arithmetic & Logical Instructions

Lecture 3
22 Mar 07

James Goodman

Department of Computer Science

Errata: NOT on the Alpha

The Alpha has no NOT instruction. I incorrectly stated that it could be synthesized with the XOR instruction, using register $31 to supply a zero operand. That was incorrect. It can be synthesized with the NOR (Alpha calls this ORNOT) instruction using register $31:

```
not A, B ≡ ornot A, $31, B
```

Alternatively, the XNOR instruction can be used (Alpha calls this operation XORNOT, but calls the instruction EQV):

```
not A,B ≡ eqv A, $31, B
```

Recommended Readings

- These notes (only after the lecture):
  http://www.cs.auckland.ac.nz/compsci210s1t/lectures
- Dr. Bruce Hutton's lecture notes:
  http://www.cs.auckland.ac.nz/compsci210s1t/resources
- Today's lecture mostly based on chapter 2 of Dr. Hutton's notes.
- You are responsible for the first 13 chapters of Dr. Hutton's notes.
  – However, if I don't talk about it in class, it probably won't be on the exam!

The Instruction/Execution Cycle

```
Do forever {
   Fetch instruction into IR from memory address in IP
   Update IP for next instruction
   Decode instruction
   Evaluate addresses
   Fetch operands from memory
   Store result
}
```

The Instruction/Execution Cycle:

Variant for Control Instructions

```
Do forever {
    Fetch instruction into IR from memory address in IP
    Update IP for next instruction
    Decode instruction
    Evaluate test criterion
    If success, store new address to PC
}
```

A Simple Program

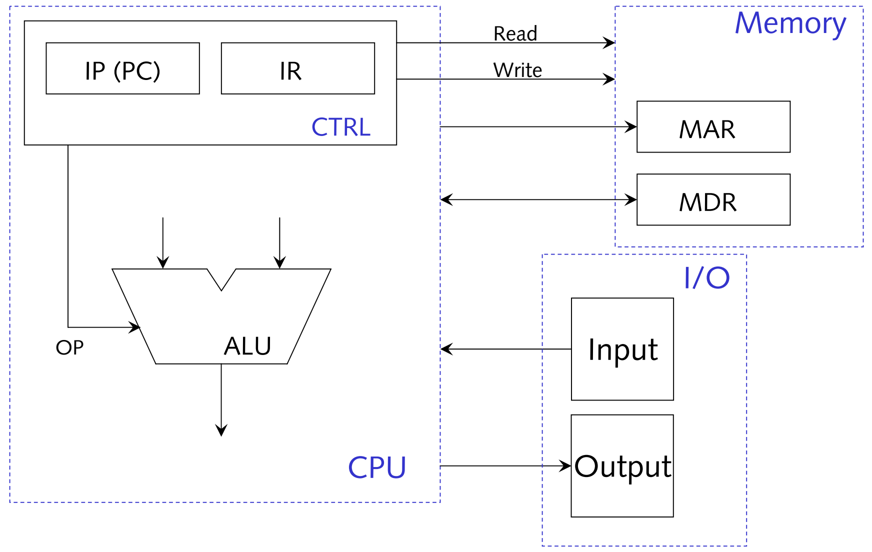
Instructions:

```
L1:   add   VA, VB, VA

L2:   sub   VC, VD, VC

L3:   mul   VC, VE, VE

L4:   bne   VA, VC, L1

L5:   halt
```

Initial values:

```
VA: 0 ⟶ 1 ⟶ 2
VB: 1
VC: 6 ⟶ 4 ⟶ 2
VD: 2
VE: 5 ⟶20 ⟶80

IP: L1→L2→L3→L4→L1→L2→L3→L4→L5
```

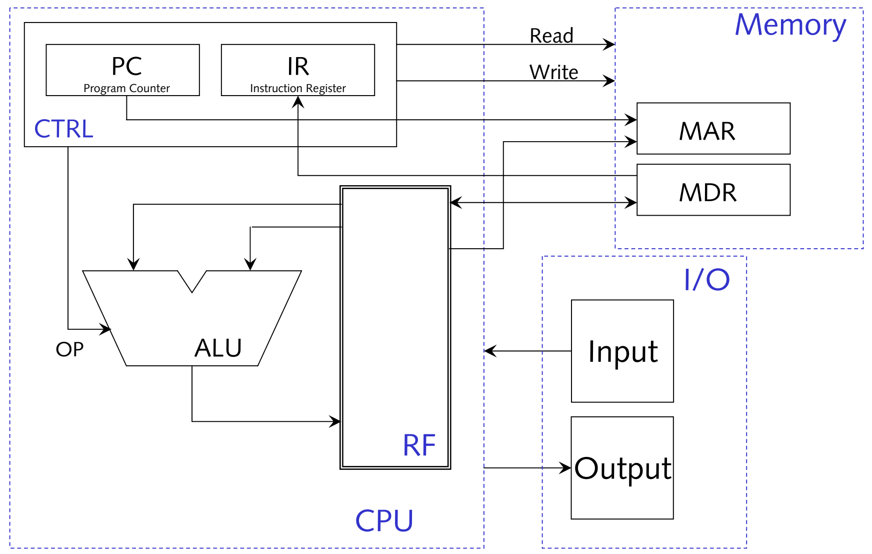
The Von Neuman Computer



The von Neuman Model

- Computer consists of CPU, Memory, I/O
- Memory may contain instructions or data (or meta-data)
- Does only one thing: the Instruction/Execution cycle

The Alpha Computer



Registers

- 32 registers
- \$0 - \$31; also names
- \$31 is special
  - when read, gives zero
  - writing has no effect

Four Categories of Instructions

- Arithmetic/Logical
  - Arithmetic
  - Logical
  - Shift
  - Compare
- Control
  - Branch on condition
  - Jump
    - Jump and link
- Memory: Load & Store
- Special

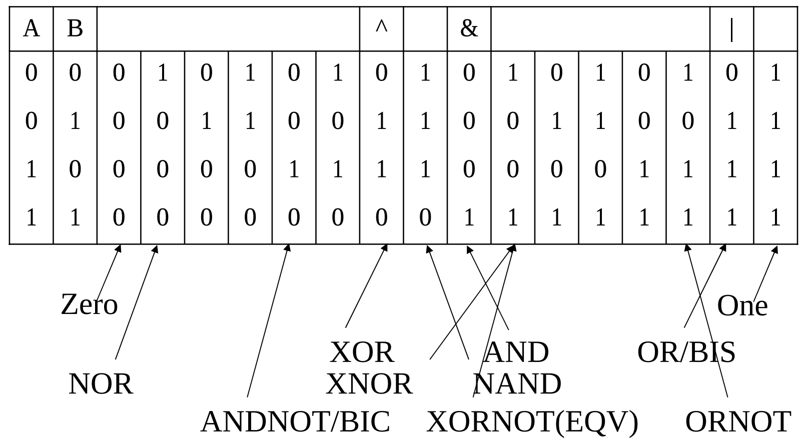
Arithmetic Instructions

- add, sub, mul (no divide)
- two sources, one destination (can be common)
- Form: **`add A,B,C`**
  - B can be an immediate, i.e., value contained in the instruction.
- Two operand types
  - Long word (32 bits): addl, subl, mull
  - Quad word (64 bits): addq, subq, mulq
- Overflow
  - Addition & subtraction: only one bit
  - Multiplication: up to 31 bits (additional multiplication ops)

Logical Instructions

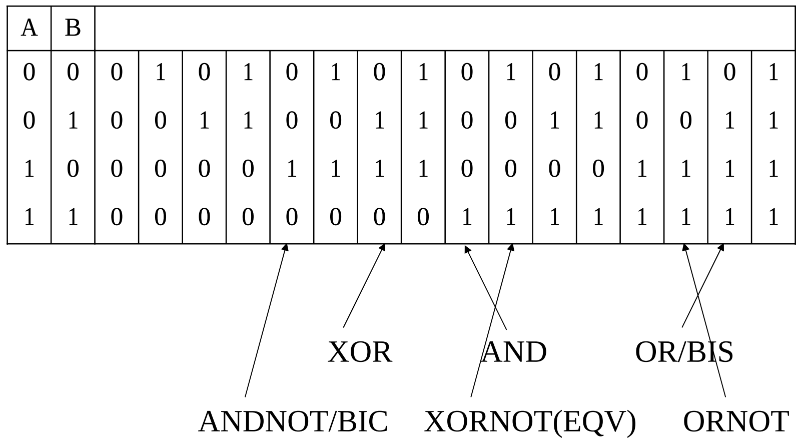
- Two sources, one destination
- Form: **`and A,B,C`**
  - B cannot be an immediate, i.e., contained in the instruction.
- One operand type: 64 bits
- Overflow: none

Boolean Functions of 2 Variables



A	B							^		&						|	
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Alpha Logical Operations



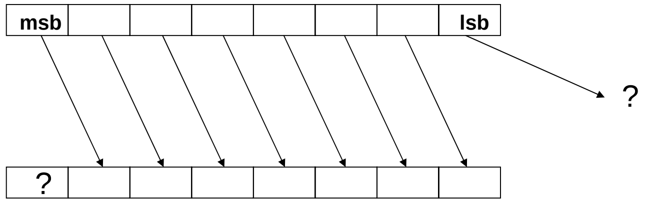
A	B																
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Shift Operations

- Form: **`sll A,Count,B`**
- A count of *i* is equivalent to *i* shifts by 1 place.
- There are three types of Shift Operations
  - logical
  - arithmetic
  - rotate

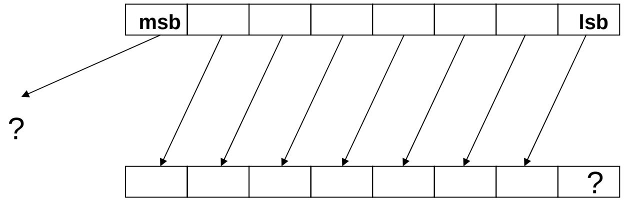
Shift Operations

- Basic Right Shift Operation:



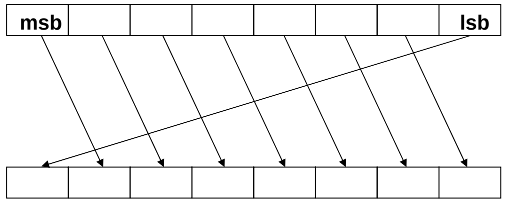
Shift Operations

- Basic Left Shift Operation:



Shift Operations

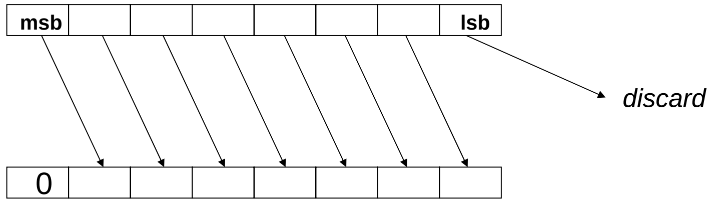
- Right Rotate Operation:



- No information lost
- For N-bit word, rotate right *N* positions has no effect
- Rotate right *i* positions is same as rotate left *N* – *i* positions
- Not implemented in Alpha (why not?)

Logical Shift Operations

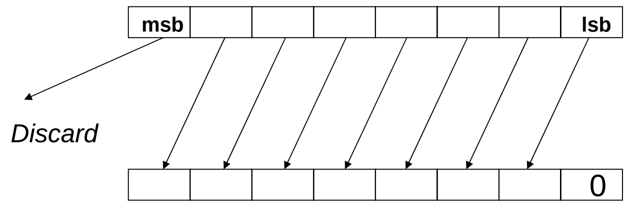
- Right Logical Shift Operation:



- Alpha instruction: **`srl`**
- Java equivalent: **>>>**

Logical Shift Operations

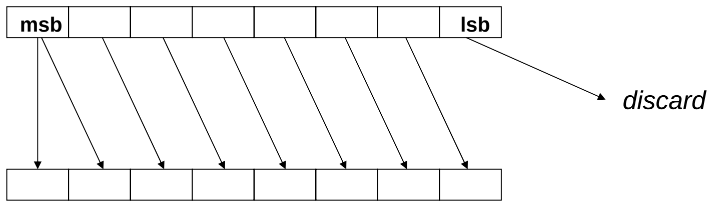
- Left Logical Shift Operation:



- Alpha instruction: **`sll`**
- Java equivalent: **`<<`**

Arithmetic Shift Operations

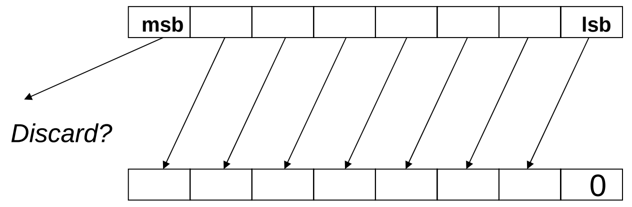
- Right Arithmetic Shift Operation
  - Unsigned integer division by power of 2



- Round down (more negative)
- Alpha instruction: **`sra`**
- Java equivalent: **`>>`**

Arithmetic Shift Operations

- Left Arithmetic Shift Operation
  - Unsigned integer multiplication by power of 2



- Overflow if MSB changes

Same as logical left shift!

- Alpha instruction: **`sll`** (no **`sla`**)
- No Java equivalent either