Computer Science 210

Computer Systems 1

2007 Semester 1

Lecture Notes

Control Instructions

Lecture 4
23 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
```

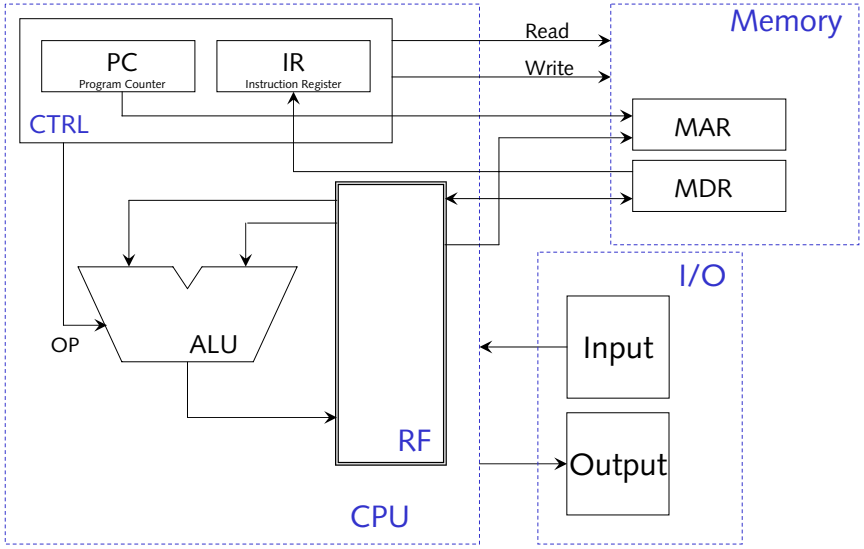
Alternatively, the XNOR instruction can be used (Alpha calls this operation XORNOT, but calls the instruction EQV):

```
not A,B ≡ eqv A, $31, B
```


Recommended Readings

- Today's lecture mostly based on chapter 2 of Dr. Hutton's notes.

The Alpha Computer



Four Categories of Instructions

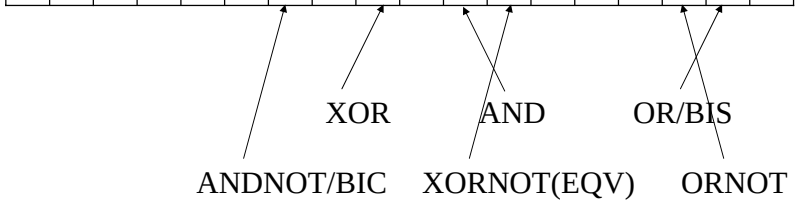
- Arithmetic/Logical
  - Arithmetic
  - Logical
  - Shift
  - Compare
- Control
  - Branch on condition
  - Jump
    - Jump and link
- Memory: Load & Store
- Special

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

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

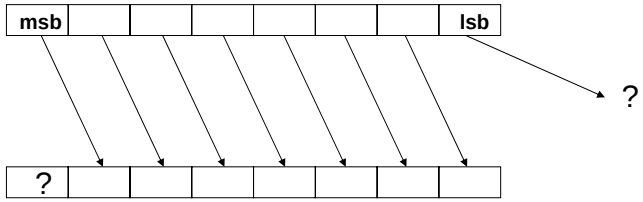


Shift Operations

- 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

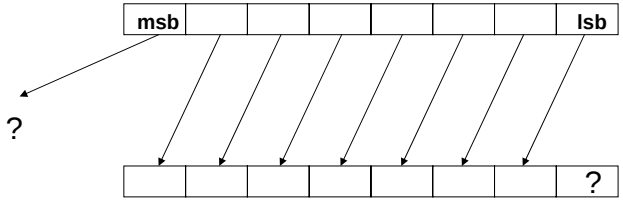
Shift Operations

- Basic Right Shift Operation:



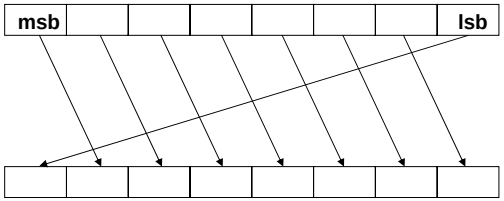
Shift Operations

- Basic Left Shift Operation:



Shift Operations

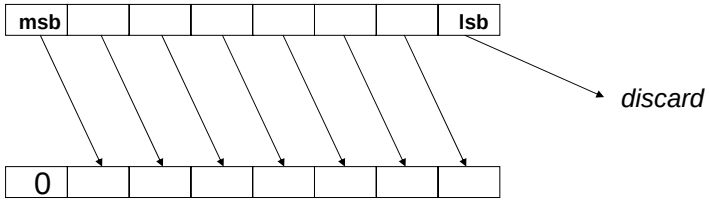
- Right Rotate Operation:



- 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

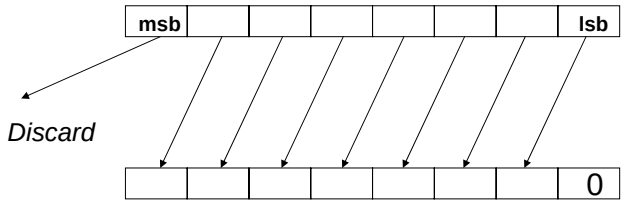
- Right Logical Shift Operation:



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

Logical Shift Operations

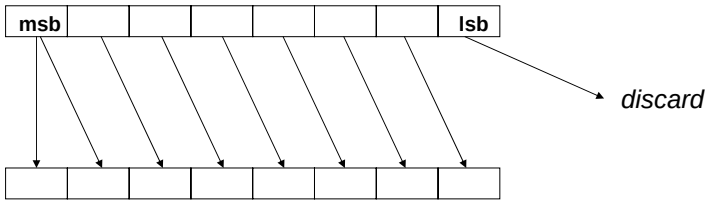
- Left Logical Shift Operation:



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

Arithmetic Shift Operations

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



- Round down (toward negative infinity)
- Alpha instruction: **`sra`**
- Java equivalent: **`>>`**
  - *same as integer division by power of 2???*

Homework: What is -5/2 in Java?

C: "… in GNU C the '/' operator always rounds towards zero. But in other C implementations, '/' may round differently with negative arguments."

-- http://www.gnu.org/software/libc/manual/html_node/Integer-Division.html

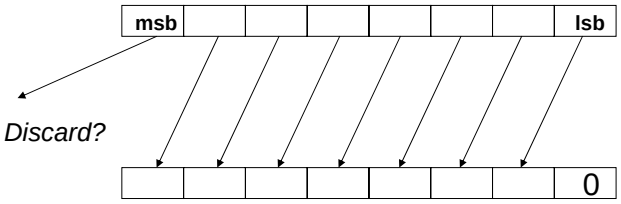
Java: "Integer division rounds toward 0."

-- http://java.sun.com/docs/books/jls/third_edition/html/expressions.html#15.17.2

Conclusion: In Java, '**`>>`**' is not the same operation as '**`/2`**^{i}'

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`**)
- Java equivalent: '**`* 2`**^{i}'

Control Instructions

Basic instruction for choosing alternate instruction path:

- Branch on condition (Kiwi-1): **`bne VA,VC,L1`**
- Alpha: **`bne a, L1`**
  - Register tested against zero
- Possible tests
  - beq : a = 0 ?
  - bne : a ≠ 0 ?
  - bge : a ≥ 0 ?
  - bgt : a > 0 ?
  - ble : a ≤ 0 ?
  - blt : a < 0 ?
  - jmp : Unconditional

Implementing Control Structures

- While loop
- If-then-else
- For loop

While Loop

Java:

```
while (a>0) {
      …
      a--;
}
```

Alpha Assembly:

```
WHILE: ble a, AFTERLOOP
       …
       sub a, 1, a
       jmp WHILE
       …
```

Test for FALSE

If-Then

Java:

```
if (a>0) {
      …
}
```

Alpha Assembly:

```
IF_THEN:
        ble a, Continue
        …
Continue:
        …
```

Test for FALSE

If-Then-Else

Java:

```
if (a>0) {
      …
} else {
      …
}
```

Alpha Assembly:

```
IF_THEN_ELSE:
          ble a, Else
          …
          jmp Continue
Else:
          …
Continue:
          …
```

Test for FALSE

For Loop

Java:

```
for (int i=0;i<10;i++) {
    …
}
```

Alpha Assembly:

```
For:
     add $31, 10, Limit
     add $31, 0, i
Loop:
     sub Limit, i, Test
     ble test, Continue
     ...
     add i, 1, i
     jmp Loop
Continue:
     …
```


Problems with For Loop Code

Alpha Assembly:

```
For:
     add $31, 10, Limit
     add $31, 0, i
Loop:
     sub Limit, i, Test
     ble test, Continue
     ...
     add i, 1, i
     jmp Loop
Continue:
     …
```

Could get overflow!

An Alternate Control Structure

- "Arithmetic" instruction Compare:
  - **`cmpeq a,b,result`** if (a=b) result=1 else result = 0
  - **`cmplt a,b,result`** if (a < b) result=1 else result = 0
  - **`cmple a,b,result`** if (a>b) result=1 else result = 0
- Additional Conditional Branch instruction
  - **`blbs result, L1`** if (low bit of result=1) jump to L1
  - **`blbc result, L1`** if (low bit of result=0) jump to L1

Additional Control Instruction

- Additional Conditional Branch Instruction
  - **`blbs a, L1`** if (low bit of a=1) jump to L1
  - **`blbc a, L1`** if (low bit of a=0) jump to L1