Logical Operations in SASM

September 20, 2000

CSC201 Section 002

Fall, 2000

Fixed Operand Lengths

- Doublewords = exactly 32 bits, bytes = exactly 8 bits
- Doesn't matter how many bits we need; this is all we get!

Shift Operations

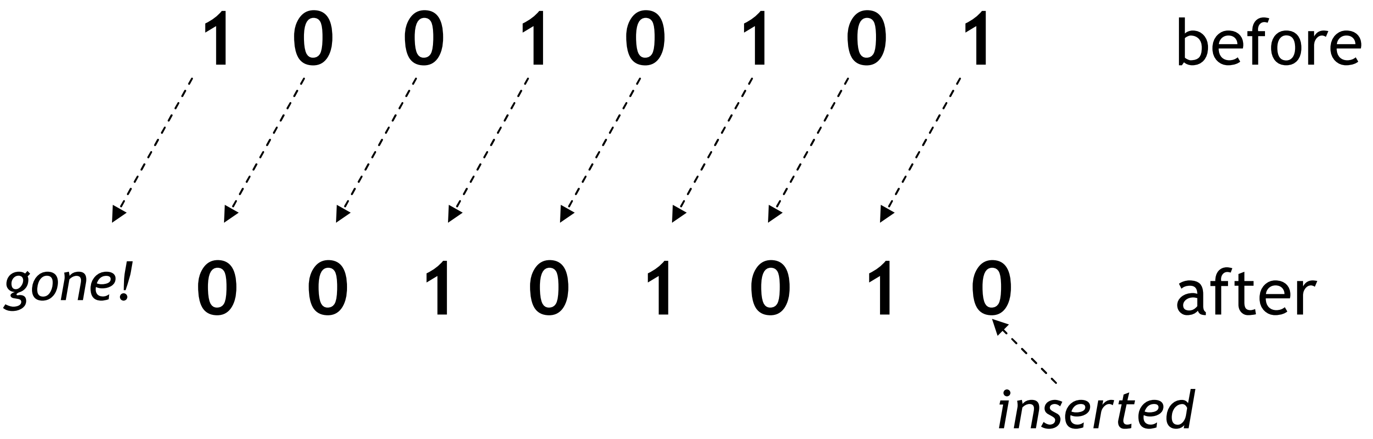
- Logical Shift
  - left and right
- Arithmetic Shift
  - (left) and right
- Rotate
  - (left) and right
- The amount (number of positions) of shifting
- *ZF and SF: affected by all except rotate*

Uses of Shift and Rotate

- Setting individual bits
- Testing individual bits
- "Cheap" divide by 2 and multiply by 2

LEFT Logical Shift

- "llsh x" -- left shift by 1 position
- Zero fill on right



1 0 0 1 0 1 0 1 before

gone! 0 0 1 0 1 0 1 0 after

inserted

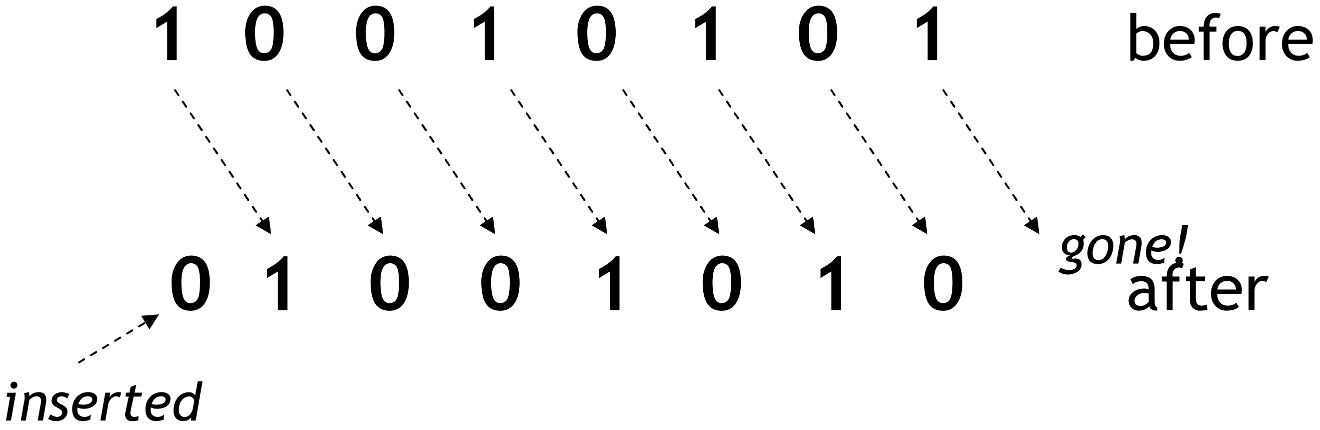
- Example: (unsigned) multiply by 2
  - overflow possibility

	Unsigned Binary	Decimal
before	0101	5
after	1010	10

	Unsigned Binary	Decimal
before	1010	10
after	0100	4???

RIGHT Logical Shift

- "rlsh x" -- right shift by 1 position
- Zero fill on left



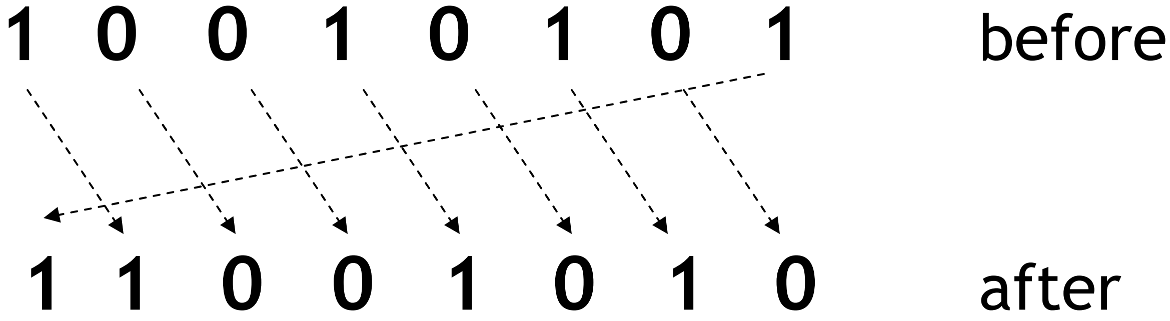
- Example: (unsigned) divide by 2

	Unsigned Binary	Decimal
before	1010	10
after	0101	5

	Unsigned Binary	Decimal
before	0101	5
after	0010	2

RIGHT Rotate

- “rrot x" -- right rotate by 1 position
- No zero fill on either end



1 0 0 1 0 1 0 1 before

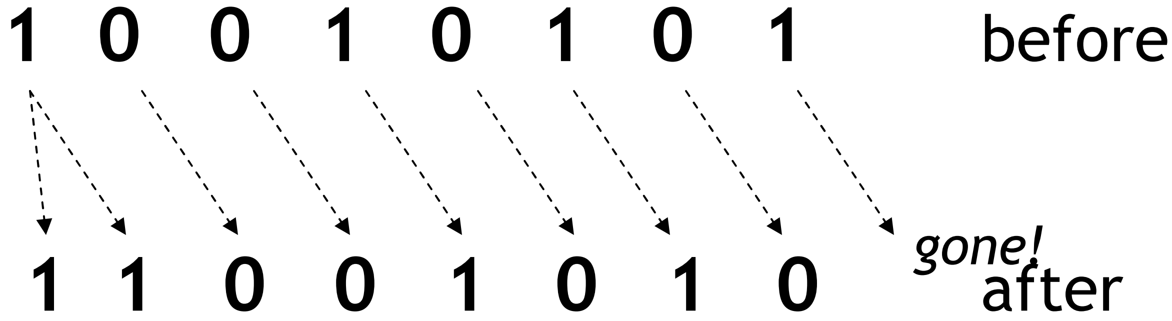
1 1 0 0 1 0 1 0 after

- Used to avoid losing any bits of the original operand (I.e., no zero fill)

	Bit String
before	0011
after	1001

RIGHT *Arithmetic* Shift

- "rash x" -- right shift by 1 position
- *Sign-extend* (sign duplicate) on the left



1 0 0 1 0 1 0 1 before

1 1 0 0 1 0 1 0 after

gone!

- Use *rash* to shift (divide by 2) *signed* twos-complement numbers, rlsh to shift unsigned numbers

RIGHT *Arithmetic* Shift

- Example: (*signed*) divide by 2

	Signed Binary	Decimal
before	1010	-6
after	1101	-3

	Signed Binary	Decimal
before	1101	-3
after	1110	-2

	Signed Binary	Decimal
before	0111	+7
after	0011	+3

SASM Logical Operations

- All affect SF and ZF, except "lnot" (no effect)
- Operands are doublewords
- "lnot dest“
    - not the same thing as "ineg"!

Operation	Operand1	Result
lnot	01011000	10100111
ineg	01011000	10101000

SASM Logical Operations

- "land dest,src"

Operation	Operand1	Operand2	Result
land	01010011	00001111	00000011

- "lor dest,src"

Operation	Operand1	Operand2	Result
lor	01010011	00001111	01011111

- "lxor dest,src"

Operation	Operand1	Operand2	Result
lxor	01010011	00001111	01011100

Selective Clearing

- "Clearing" to 0 by AND'ing with 0
  - AND'ing with 1 = no effect
- "Masking": 0's where clearing desired, 1's elsewhere

Operation	Operand1	Mask	Result
land	01010011	00001111	00000011

Operation	Operand1	Mask	Result
land	01010011	00111100	00010000

Selective Setting

- "Setting" to 1 by OR'ing with 1
  - OR'ing with 0 = no effect
- "Masking": 1's where setting desired, 0's elsewhere

Operation	Operand1	Mask	Result
lor	01010011	00001111	01011111

Operation	Operand1	Mask	Result
lor	01010011	00111100	01111111

Inserting Bits

- "Inserting" a new value into specific bits

Operation	Operand1	Bits to Insert	Result
"inserting"	01010011	xxx0101x	01001011

- Step 1: selectively clear

Operation	Operand1	Mask 1	Result 1
land	01010011	11100001	01000001

- Step 2: selectively set using result from step 1

Operation	Operand2	Mask 2	Result 2
lor	01000001	00001010	01001011

Testing Bits

- AND with 1 in the bit position you want to test; zeros elsewhere
  - Result is 0 if bit was cleared
  - Result is 1 if bit was set
  - Test result using “bez” or “bnz”

```
        land    op1, 00000001h
        bez     lsb_was_cleared
        ...
```

What will bcnt be after…

```
          move  icnt, 0
          move  bcnt, 0
Forloop:
          compare icnt, 32
          bez   endfor
          rrot  bstr
          move  op1, bstr       ; copy bstr
                                ; before you
                                ; test it
          land  op1, 00000001h
          bez   lsb_was_cleared
          iadd  bcnt, 1
Lsb_was_cleared:
          iadd  icnt, 1
          br    forloop
Endfor: …
```