Basic Pentium Instructions

October 18

CSC201 Section 002

Fall, 2000
The EFLAGS Register

- Bit 11 = Overflow Flag
- Bit 7 = Sign Flag
- Bit 6 = Zero Flag
- Bit 0 = Carry Flag
- "Sets the flags" means sets OF, ZF, SF, CF in the "normal" way
Operand Types and Notation

• Register operands
  - reg (= reg32), reg8

• Memory operands
  - mem (= mem32), mem8
  - Any memory addressing mode allowed unless specified otherwise

• Immediate values
  - imm (=imm32), imm8
"mov" Instruction

- Operand combinations: reg/mem, reg/imm

  "mov EAX, EBX"
  "mov EAX, 0A34H"
  "mov [EBP+4], EAX"
  "mov myvar, 022334455h"

- Another combination: reg, mem

  "mov EAX, myvar"

- 8-bit versions of all of the above
- No effect on flags
"movsx" and "movzx" Instructions

- Operands combination: reg, reg8/mem8
  
  "movsx EAX, BL"
  "movsx EAX, mychar1"

- No effect on flags
"lea" Instruction

- Operand combination: reg, mem

  "lea EBP, myvar"
  "lea EBP, [myvar+5]"

- No effect on flags
Push and Pop

• Push
  - Operand combination: reg/mem/imm
    “push EAX”
    “push [ESI+4]”
    “push 22AFFF33h”
  - Can also push a byte-length immediate

• Pop
  - Operand combination: reg/mem
    “pop EAX”
    “pop myvar”
  - Cannot pop byte-length operands
"add" and "sub" Instructions

• Operands: reg/mem, reg/imm

  "add  myvar, ECX"
  "sub  EBX, -22"

• Another combination: reg, mem

  "add  EAX, myvar"

• 8-bit versions of the above

• Setting of the flags...
  - Overflow detect (OF) assumes operands are twos-complement numbers
  - Unsigned overflow detect: use CF instead
"cmp" Instruction

- Same as "sub", but result value not stored in first operand
  - Only sets the flags
"inc" and "sub" Instructions

• Operand combination: reg/mem
  - Like "add operand, 1" and "sub operand, 1"

• 8-bit versions of the above

• Sets the flags
  - Overflow detect assumes operand is twos-complement number
  - Does not affect CF; unsigned overflow detect not possible
"neg" Instruction

- Two's-complement a number
- Operands: reg / mem

"neg EAX"

- 8-bit versions of the above
- Sets the flags
"mul" Instruction

- *Unsigned* multiply
- **Operand combination: reg/mem**
  - **Meaning:** $EDX:EAX \leftarrow EAX * \text{reg (or mem)}$

```
"mul EBX"
"mul mydw1"
```

- *(There is an 8-bit version we will not use)*
- **Only useful flag:** $CF = 0$ means $DX = 0$, otherwise $CF = 1$
"imul" Instruction

- *Signed* multiply
- One-address, two-address, and three-address formats!
- Operands: reg/mem
  - EDX:EAX <-- EAX * reg (or mem)
  - "mul EBX"

- Operands: reg, reg/mem
  - reg <-- reg * reg (or mem)
  - "mul EAX, myvar"
"imul" Instruction

- Operand combination: reg, reg/mem, imm
  - reg <-- reg (or mem) * imm

  “mul EAX, myvar, 55”

- (There is an 8-bit version we will not use)
- Only useful flag: CF = 0 means result fits in EAX alone (or destination register), CF = 1 otherwise
"div" Instruction

• *Unsigned* divide

• Operand combination: reg/mem
  - EDX <-- remainder of EDX:EAX / reg (or mem), EAX <-- quotient

  “div EBX”

• (There is an 8-bit version we will not use)

• No effect on flags, but check for divide by zero (or overflow) possibility first!
  - Will generate an exception (program halts) if you don't catch it
"idiv" Instruction

- *Signed* divide
- Otherwise, same as "div"
"not" Instruction

- operand: reg/mem

"not [EAX+35]"

- 8-bit version also
- no effect on flags
"or", "and", "xor", and "test" Instructions

• operands: reg/mem, reg/imm
  
  "or EAX, EDX"

• reg, mem
  
  "or EAX, myvar"

• 8-bit versions also

• affects SF and ZF (not CF or OF)
  - "test" only affects the flags, no result stored

• Note: xor'ing a value with itself clears it to 0; programming trick!
"shr", "sar", "shl" and "sal"
Instructions

• operands: reg/mem, cl/imm8

```

  "sar  dword ptr [EBP - 8], 1"
  "shr  EAX, CL"

```

• cl and imm8 should have a value between 0 and 31

• shr does zero fill, sar does sign extension

• shl and sar do same thing

• 8-bit versions also (but imm8 can only = 1)

• affects ZF, SF, and CF
  - shr and sar: CF = least-significant-bit (before shifting)
  - shl and sal: CF = most-significant-bit (before shifting)
"rol" and "ror" Instructions

- operands: reg/mem, cl/imm8

  "ror  dword ptr [EBX], 4"
  "rol  EAX, CL"

- cl and imm8 should have a value between 0 and 31

- 8-bit versions also (but imm8 can only = 1)

- affects ZF, SF, and CF
  - ror: CF = least-significant-bit (before shifting)
  - rol: CF = most-significant-bit (before shifting)