Assembly Control Flow

- Unconditional Jump –
  \[ \text{jmp~[label]} \]
  e.g.,
  \[ \text{jmp~L1} \]
  ...
  L1: target instruction
  ...
  \[ \text{jmp~*%eax~indirect,~dest~in~%eax} \]
  \[ \text{jmp~*(%eax)~indirect,~dest~in~M[\%eax]} \]

Conditional Control Flow

- In x86, separate expression eval and cond branch inst.
- Compare –
  \[ \text{cmpl~b,~a} \]
  \( r,m,i \), \( r,m \)
- Perform operation temp = a – b, throw away temp and set flags based on results of subtraction operation.
- Flags can also be set as a result of normal arithmetic and/or logical operations.

Conditional Jumps

- In x86, separate expression eval and cond branch inst.
- Compare –
  \[ \text{cmpl~b,~a} \]
  \( r,m,i \), \( r,m \)
- Perform operation temp = a – b, throw away temp and set flags based on results of subtraction operation.
- Flags can also be set as a result of normal arithmetic and/or logical operations.

General Conditionals

- \( \text{JZ~zero~alternate~name~JE} \)
- \( \text{JNZ~not~zero~alternate~name~JNE} \)
- \( \text{JC~carry} \)
- \( \text{JNC~no~carry} \)

Signed Conditionals

- \( \text{JG~greater~than~(a~>~b)~alternate~name~JNLE} \)
- \( \text{JL~less~than~(a~<~b)~alternate~name~JNGE} \)
- \( \text{JGE~greater~or~equal~(a~\geq~b)~alternate~name~JNL} \)
- \( \text{JLE~less~or~equal~(a~\leq~b)~alternate~name~JNG} \)
Unsigned Conditionals

JA  above (a > b)  alternate name JNBE
JB  below (a < b)  alternate name JNAE
JAE above or equal (a ≥ b) alternate name JNB
JBE below or equal (a ≤ b) alternate name JNA

Loop Instruction

```assembly
loop      [label]
e.g.,
movl    $15, %ecx  /* # of iterations */
.L1     .
    .
    loop    .L1
```
• Operation is as follows:
  – decrement %ecx
  – test %ecx for zero, if not zero jump to [label]

Control Flow in C

```c
if ... then ...
e.g.,
if ([cond expr]) {
    if (var1 > var2) {
        [true body]
        var1 = var1 + var2;
    }
    [main body]
} [main body]

if ... then
```
if ... then ... else

```c
if (var1 > var2) {
    var1 = var1 + var2;
    var2 = 0;
} else {
    var2 = var2 + var1;
    var1 = var2;
}
```

```asm
movl var1, %eax
cmp  var2, %eax
jle  false_body
addl var2, %eax
movl $0, var2
jmp  main_body
```

```
false_body:
    %eax, var2
    movl %eax, var1
    main_body: ...
```

Conditional if ... then ... else

```c
if (((cond1) && (cond2)) || (cond3)) {
    [true body]
} else {
    [false body]
}
```

```asm
if (((cond1) && (cond2)) || (cond3))
```
while loop

while ([cond expr]) {
    [loop body]
} 
[main body]

• Assembly
  
  while_loop:
  
  cmp [cond expr oper]
  j[!cond] exit_while
  [loop body]
  jmp while_loop
  exit_while:
  [main body]

Notes for Java Programmers

• Declare index variable before for loop
  
  int i;
  for (i=0; i<n; i++) {
  }

  vs.
  
  for (int i=0; i<n; i++) {

• Uninitialized variables
  
  int main (int argc, char* argv[]) {
      int i;
      factorial(i);
      return 0;
  }

Notes for Java Programmers

• Error handling:
  
  – No exceptions, must look at return values
  – E.g.,
    
    int open_file(filename) {
      /* attempt to open filename */
      if (error) return -1;
      else return 0;
    }

Command line arguments in C

int main(int argc, char * argv[]) {

  • argc
    
    – Number of arguments (including program name)
    
    • argv
      
      – Array of char*s (that is, an array of 'C' strings)
      
      – argv[0]: = program name
      – argv[1]: = first argument
      – ...
      – argv[argc-1]: last argument

Program echoargs.c

#include <stdio.h>

int main(int argc, char* argv[]) {

  int i;

  printf("%d arguments\n", argc);
  for(i = 0; i < argc; i++)
    printf(" %d: %s\n", i, argv[i]);
  return 0;
}

> ./echoargs Humankind cannot stand very much reality
7 arguments
0: ./echoargs
1: Humankind
2: cannot
3: stand
4: very
5: much
6: reality
>