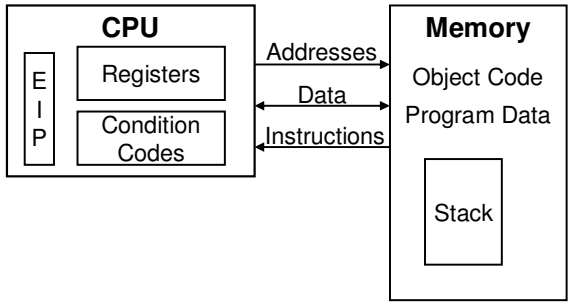
15213 Recitation Section C

Shimin Chen
Sept. 16, 2002

Outline

- Assembly Review
- C ⇆ ASM using GDB
- ASM ⇆ C

Assembly Review: Machine Model



Assembly Format

- Op Src, Dest
  - add %eax, %ebx # %ebx += %eax
  - sub %eax, %ebx # %ebx -= %eax
- Op Arg
  - jmp 0x87654321 # unconditional branch
  - jge 0x87654321 # branch if >= in signed # comparison

Memory Addressing Mode

- Generic form:
  - D(R1, R2, S)
  - Address: Reg[R1] + Reg[R2]*S + D
  - e.g. 0x8(%eax, %ebx, 0x4)
  - the address is %eax + %ebx * 0x4 + 0x8
- Special forms:
  - omit D, R1, R2, or S
  - (R1), D(R1), (R1, R2), D(R1, R2)

Exercise: What do the ASM mean?

1) sub %ecx, %edx
2) cmp %ecx, 0x4
   jge 0x12345678
3) mov (%ebx), %eax
4) mov (%ebx, %esi, 0x4), %edi
5) lea (%ebx, %esi, 0x4), %edi
6) xor %ecx, %ecx

Procedure Related Instructions

int a_func (int arg1, int arg2, int arg3)

- Get arguments:
  - arg1: mov **8(%ebp)**,%ecx
  - arg2: mov **12(%ebp)**,%ecx
  - arg3? mov **16(%ebp)**,%ecx
- Set return value:
  - mov 0x1, %eax # return 1;

C˪ ASM

- Compilation and GDB basics
- C˪ ASM Examples

Compiling and Debugging C Code

- Generating ASM with gcc
  - gcc –O –S -Wall example.c
  - generate example.s
- Debugging C code
  - gcc –O –g –o example -Wall example.c
  - gdb example

What if compiling without "-g"?

- *gcc –O –o example -Wall example.c*
- gdb will not know the C code for assembly
- *the same as in L2 "bomb lab"*
- use gdb to examine the object code
  - Other tools (objdump etc.) see L2 description

Example: func1

```
int func1(int a, int b)
{
  int x, y;

  x = a + b;
  y = 2*x - b;

  return x*y;
}
```


ASM of func1

```
Dump of assembler code for function func1:
0x8048420 <func1>:      push   %ebp
0x8048421 <func1+1>:    mov    %esp,%ebp
0x8048423 <func1+3>:    mov    0xc(%ebp),%eax
0x8048426 <func1+6>:    mov    0x8(%ebp),%ecx
0x8048429 <func1+9>:    add    %eax,%ecx
0x804842b <func1+11>:   lea    (%ecx,%ecx,1),%edx
0x804842e <func1+14>:   sub    %eax,%edx
0x8048430 <func1+16>:   mov    %ecx,%eax
0x8048432 <func1+18>:   imul   %edx,%eax
0x8048435 <func1+21>:   mov    %ebp,%esp
0x8048437 <func1+23>:   pop    %ebp
0x8048438 <func1+24>:   ret
0x8048439 <func1+25>:   lea    0x0(%esi),%esi
End of assembler dump.
```


ASM of func1

```
Dump of assembler code for function func1:
0x8048420 <func1>:      push   %ebp
0x8048421 <func1+1>:    mov    %esp,%ebp
0x8048423 <func1+3>:    mov    0xc(%ebp),%eax      #%eax=b
0x8048426 <func1+6>:    mov    0x8(%ebp),%ecx      #%ecx=a
0x8048429 <func1+9>:    add    %eax,%ecx           #%ecx=a+b
0x804842b <func1+11>:   lea    (%ecx,%ecx,1),%edx  #%edx=2*%ecx
0x804842e <func1+14>:   sub    %eax,%edx           #%edx-=b
0x8048430 <func1+16>:   mov    %ecx,%eax           #%eax=x
0x8048432 <func1+18>:   imul   %edx,%eax           #return x*y
0x8048435 <func1+21>:   mov    %ebp,%esp
0x8048437 <func1+23>:   pop    %ebp
0x8048438 <func1+24>:   ret
0x8048439 <func1+25>:   lea    0x0(%esi),%esi
End of assembler dump.
```

Using GDB to run the program

- Let's use gdb to run the program and examine registers and memory locations
- break func1
- run
- p/x $ebp
- x/2wx $ebp+8

Example 2

```
int func2(int a, int b)
{
  if(a>b)
    return a;
  else
    return b;
}
```


ASM of func2

```
Dump of assembler code for function func2:
0x804843c <func2>:      push   %ebp
0x804843d <func2+1>:    mov    %esp,%ebp
0x804843f <func2+3>:    mov    0x8(%ebp),%edx
0x8048442 <func2+6>:    mov    0xc(%ebp),%eax
0x8048445 <func2+9>:    cmp    %eax,%edx
0x8048447 <func2+11>:   jle    0x804844b <func2+15>
0x8048449 <func2+13>:   mov    %edx,%eax
0x804844b <func2+15>:   mov    %ebp,%esp
0x804844d <func2+17>:   pop    %ebp
0x804844e <func2+18>:   ret
0x804844f <func2+19>:   nop
End of assembler dump.
```


ASM of func2

```
Dump of assembler code for function func2:
0x804843c <func2>:      push   %ebp
0x804843d <func2+1>:    mov    %esp,%ebp
0x804843f <func2+3>:    mov    0x8(%ebp),%edx          #%edx=a
0x8048442 <func2+6>:    mov    0xc(%ebp),%eax          #%eax=b
0x8048445 <func2+9>:    cmp    %eax,%edx               #%edx<=%eax?
0x8048447 <func2+11>:   jle    0x804844b <func2+15> #
0x8048449 <func2+13>:   mov    %edx,%eax               #%eax=a
0x804844b <func2+15>:   mov    %ebp,%esp
0x804844d <func2+17>:   pop    %ebp
0x804844e <func2+18>:   ret
0x804844f <func2+19>:   nop
End of assembler dump.
```

Example 3

```
int func3(int a, int b)
{
    int r = 0xDEADBEEF;
    switch(a) {
    case 0:
    case 1:
            r = b; break;
    case 2: r = a+b; break;
    case 3: r = a-b; break;
    case 4: r = a*b; break;
    default:;
    }
    return r;
}
```


ASM of func3

```
Dump of assembler code for function func3:
0x8048450 <func3>:     push   %ebp
0x8048451 <func3+1>:   mov    %esp,%ebp
0x8048453 <func3+3>:   mov    0x8(%ebp),%edx
0x8048456 <func3+6>:   mov    0xc(%ebp),%ecx
0x8048459 <func3+9>:   mov    $0xdeadbeef,%eax
0x804845e <func3+14>:  cmp    $0x4,%edx
0x8048461 <func3+17>:  ja     0x804848b <func3+59>
0x8048463 <func3+19>:  jmp    *0x8048598(,%edx,4)
0x804846a <func3+26>:  lea    0x0(%esi),%esi
0x8048470 <func3+32>:  mov    %ecx,%eax
0x8048472 <func3+34>:  jmp    0x804848b <func3+59>
0x8048474 <func3+36>:  lea    (%ecx,%edx,1),%eax
0x8048477 <func3+39>:  jmp    0x804848b <func3+59>
```


ASM of func3

```
0x8048479 <func3+41>:  lea    0x0(%esi,1),%esi
0x8048480 <func3+48>:  mov    %edx,%eax
0x8048482 <func3+50>:  sub    %ecx,%eax
0x8048484 <func3+52>:  jmp    0x804848b <func3+59>
0x8048486 <func3+54>:  mov    %edx,%eax
0x8048488 <func3+56>:  imul   %ecx,%eax
0x804848b <func3+59>:  mov    %ebp,%esp
0x804848d <func3+61>:  pop    %ebp
0x804848e <func3+62>:  ret

(gdb) x/5wx 0x8048598
0x8048598 <_IO_stdin_used+4>:  0x08048470  0x08048470
    0x08048474  0x08048480
0x80485a8 <_IO_stdin_used+20>:  0x08048486
```


ASM of func3

```
Dump of assembler code for function func3:
0x8048450 <func3>:     push   %ebp
0x8048451 <func3+1>:   mov    %esp,%ebp
0x8048453 <func3+3>:   mov    0x8(%ebp),%edx          #%edx=a
0x8048456 <func3+6>:   mov    0xc(%ebp),%ecx          #%ecx=b
0x8048459 <func3+9>:   mov    $0xdeadbeef,%eax        #%eax is r
0x804845e <func3+14>:  cmp    $0x4,%edx               #(a>4?)
0x8048461 <func3+17>:  ja     0x804848b <func3+59>
0x8048463 <func3+19>:  jmp    *0x8048598(,%edx,4)     #jmp table
0x804846a <func3+26>:  lea    0x0(%esi),%esi          #nop
0x8048470 <func3+32>:  mov    %ecx,%eax               #r=b
0x8048472 <func3+34>:  jmp    0x804848b <func3+59>
0x8048474 <func3+36>:  lea    (%ecx,%edx,1),%eax      #r=a+b
0x8048477 <func3+39>:  jmp    0x804848b <func3+59>
```

ASM of func3

```
0x8048479 <func3+41>:   lea    0x0(%esi,1),%esi          #nop
0x8048480 <func3+48>:   mov    %edx,%eax                 #r=a
0x8048482 <func3+50>:   sub    %ecx,%eax                 #r-=b
0x8048484 <func3+52>:   jmp    0x804848b <func3+59>
0x8048486 <func3+54>:   mov    %edx,%eax                 #r=a
0x8048488 <func3+56>:   imul   %ecx,%eax                 #r*=b
0x804848b <func3+59>:   mov    %ebp,%esp
0x804848d <func3+61>:   pop    %ebp
0x804848e <func3+62>:   ret

(gdb) x/5wx 0x8048598
0x8048598 <_IO_stdin_used+4>:   0x08048470      0x08048470
    0x08048474      0x08048480
0x80485a8 <_IO_stdin_used+20>:  0x08048486
```


Example 4

```
void func4 ()
{
        printf ("hello world!\n");
}
```


ASM of func4

```
0x8048490 <func4>:      push   %ebp
0x8048491 <func4+1>:    mov    %esp,%ebp
0x8048493 <func4+3>:    sub    $0x8,%esp
0x8048496 <func4+6>:    add    $0xfffffff4,%esp      #
0x8048499 <func4+9>:    push   $0x80485ac            #
0x804849e <func4+14>:   call   0x804833c <printf>  #  calling printf
0x80484a3 <func4+19>:   mov    %ebp,%esp
0x80484a5 <func4+21>:   pop    %ebp
0x80484a6 <func4+22>:   ret
0x80484a7 <func4+23>:   nop

(gdb) x/s 0x80485ac
0x80485ac <_IO_stdin_used+24>:   "hello world!\n"
```


ASM轉 C

```
int func5(int x)
{
        ???
}
```

ASM比 C: write C code for ASM

```
0x80483c0 <func5>:      push   %ebp
0x80483c1 <func5+1>:    mov    %esp,%ebp
0x80483c3 <func5+3>:    mov    0x8(%ebp),%ecx
0x80483c6 <func5+6>:    xor    %eax,%eax
0x80483c8 <func5+8>:    xor    %edx,%edx
0x80483ca <func5+10>:   cmp    %ecx,%edx
0x80483cc <func5+12>:   jge    0x80483d7 <func5+23>
0x80483ce <func5+14>:   mov    %esi,%esi
0x80483d0 <func5+16>:   add    %edx,%eax
0x80483d2 <func5+18>:   inc    %edx
0x80483d3 <func5+19>:   cmp    %ecx,%edx
0x80483d5 <func5+21>:   jl     0x80483d0 <func5+16>
0x80483d7 <func5+23>:   mov    %ebp,%esp
0x80483d9 <func5+25>:   pop    %ebp
0x80483da <func5+26>:   ret
0x80483db <func5+27>:   nop
```