Machine Level Programming: Basics (II)

CSCI 237: Computer Organization
12th Lecture, Wednesday, October 3

Kelly Shaw

1

1

Administrative Details

- Lab #3 assigned today. Checkpoint due Tuesday at 11pm
  - Any questions?
- Weekly quiz open today 2:30pm
  - Due Friday at 2:30pm
- Read CSAPP 3.5-3.6
- Snack and Gab
  - Today, 4:10-4:30, in CS Commons
- CS Colloquium talk Friday at 2:35pm in Wege
  - Student talks about summer

2

2

Last Time: Machine-Level Programming: Basics

- Instruction Set Architecture (ISA)
- Assembly instruction basics: registers, operands, move
- Dynamic memory allocation

3

3

Today: Machine-Level Programming: Basics

- Dynamic memory allocation
- Assembly instruction basics: registers, operands, move
- Arithmetic and logical operations

4

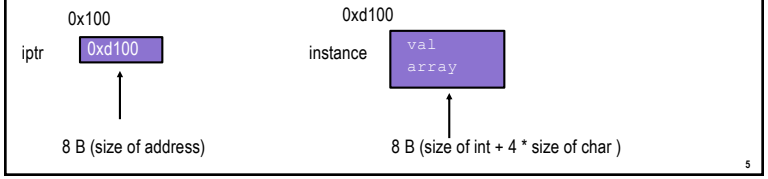
4

struct example (Static)

```
struct item {
   int val;
   char array[4];
};

struct item instance;
struct item *iptr;
iptr = &instance;

iptr->val = 10;                    // instance.val = 10;
iptr->array[3] = 'a';              // instance.array[3] = 'a';
```



5

5

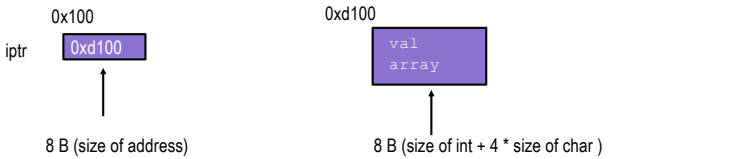
struct example

```
struct item {
   int val;
   char array[4];
};

struct item *iptr;
iptr = (struct item*)malloc(sizeof(struct item));

iptr->val = 10;
iptr->array[3] = 'a';

free(iptr);
```



6

6

Limitations of Static Memory Allocation

- Sometimes we don't know how much memory we're going to need until we read our input

```
int input_array[100000]; // Is this big enough?
```

- Sometimes we want data to persist beyond the end of a function call, but we don't want to declare global variables

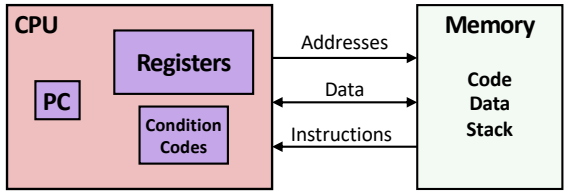
```
int * combine(int arr1[], int len1, int arr2[], int len2){
    int result[len1+len2];
    for(int i = 0; i < len1; i++)
       result[i] = arr1[i];
    for(int i = 0; i < len2; i++)
       result[len1+i] = arr2[i];

    return result;  // This won't work!
}
```

7

7

Review: Assembly/Machine Code View



Programmer-Visible State

- **PC: Program counter**
  - Address of next instruction
  - Called "RIP" (x86-64)
- **Register file**
  - Heavily used program data
- **Condition codes**
  - Store status information about most recent arithmetic or logical operation
  - Used for conditional branching
- **Memory**
  - Byte addressable array
  - Code and user data
  - Stack to support procedures

8

8

`movq` Operand Combinations

	Source	Dest	Src,Dest	C Analog
movq	*Imm*	*Reg*	`movq $0x4,%rax`	`temp = 0x4;`
		Mem	`movq $-147,(%rax)`	`*p = -147;`
	Reg	*Reg*		
		Mem		
	Mem	*Reg*		

9

9

`movq` Operand Combinations

	Source	Dest	Src,Dest	C Analog
movq	*Imm*	*Reg*	`movq $0x4,%rax`	`temp = 0x4;`
		Mem	`movq $-147,(%rax)`	`*p = -147;`
	Reg	*Reg*	`movq %rax,%rdx`	`temp2 = temp1;`
		Mem	`movq %rax,(%rdx)`	`*p = temp;`
	Mem	*Reg*	`movq (%rax),%rdx`	`temp = *p;`

Cannot do memory-memory transfer with a single instruction

10

10

Simple Memory Addressing Modes

- Normal (R) Mem[Reg[R]]
  - Register R specifies memory address
  - Aha! Pointer dereferencing in C

```
movq (%rcx),%rax
```

- Displacement D(R) Mem[Reg[R]+D]
  - Register R specifies start of memory region
  - Constant displacement D specifies offset

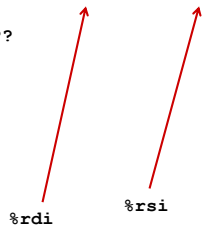
```
movq 8(%rbp),%rdx
```

11

11

Example of Simple Addressing Modes

```
void
whatAmI(<type> a, <type> b)
{
      ????
}
```




```
whatAmI:
   movq    (%rdi), %rax
   movq    (%rsi), %rdx
   movq    %rdx, (%rdi)
   movq    %rax, (%rsi)
   ret
```

12

12

Example of Simple Addressing Modes

```
void swap
   (long *xp, long *yp)
{
  long t0 = *xp;
  long t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```

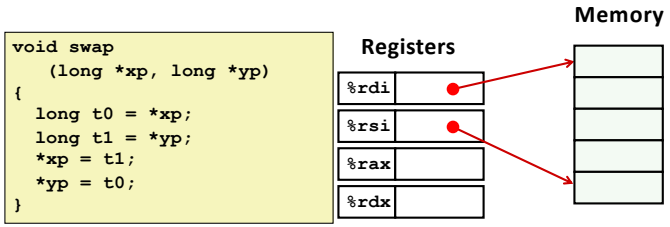
```
swap:
   movq    (%rdi), %rax
   movq    (%rsi), %rdx
   movq    %rdx, (%rdi)
   movq    %rax, (%rsi)
   ret
```

13

13

Understanding swap()

```
void swap
   (long *xp, long *yp)
{
  long t0 = *xp;
  long t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```



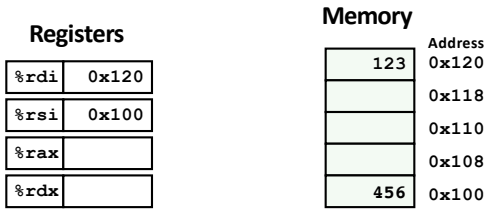
Register	Value
%rdi	xp
%rsi	yp
%rax	t0
%rdx	t1

```
swap:
   movq    (%rdi), %rax  # t0 = *xp
   movq    (%rsi), %rdx  # t1 = *yp
   movq    %rdx, (%rdi)  # *xp = t1
   movq    %rax, (%rsi)  # *yp = t0
   ret
```

14

14

Understanding swap()

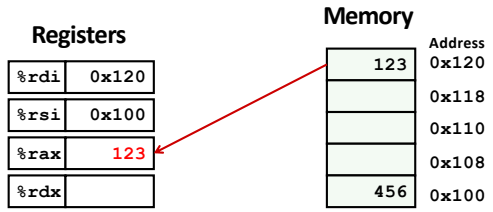



```
swap:
   movq    (%rdi), %rax  # t0 = *xp
   movq    (%rsi), %rdx  # t1 = *yp
   movq    %rdx, (%rdi)  # *xp = t1
   movq    %rax, (%rsi)  # *yp = t0
   ret
```

15

15

Understanding swap()

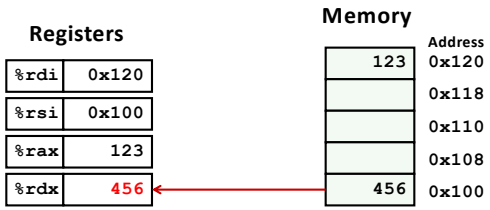



```
swap:
   movq    (%rdi), %rax  # t0 = *xp
   movq    (%rsi), %rdx  # t1 = *yp
   movq    %rdx, (%rdi)  # *xp = t1
   movq    %rax, (%rsi)  # *yp = t0
   ret
```

16

16

Understanding swap()

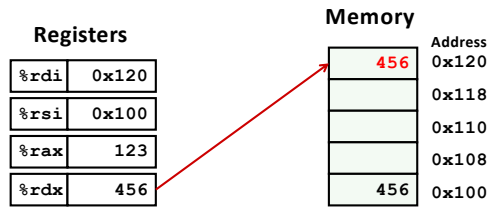



```
swap:
   movq    (%rdi), %rax  # t0 = *xp
   movq    (%rsi), %rdx  # t1 = *yp
   movq    %rdx, (%rdi)  # *xp = t1
   movq    %rax, (%rsi)  # *yp = t0
   ret
```

17

17

Understanding swap()

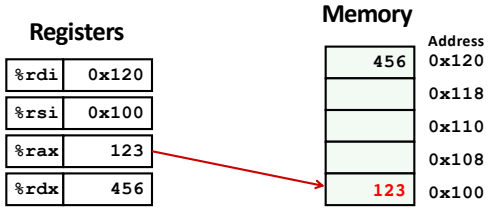



```
swap:
   movq    (%rdi), %rax  # t0 = *xp
   movq    (%rsi), %rdx  # t1 = *yp
   movq    %rdx, (%rdi)  # *xp = t1
   movq    %rax, (%rsi)  # *yp = t0
   ret
```

18

18

Understanding swap()




```
swap:
   movq    (%rdi), %rax  # t0 = *xp
   movq    (%rsi), %rdx  # t1 = *yp
   movq    %rdx, (%rdi)  # *xp = t1
   movq    %rax, (%rsi)  # *yp = t0
   ret
```

19

19

Practice On Your Own

- What would C code look like for this assembly code?

```
fcn:
   movq    $10, %rax
   movq    %rax, (%rdi)
   ret
```

20

20

Simple Memory Addressing Modes

- Normal (R) Mem[Reg[R]]
  - Register R specifies memory address
  - Aha! Pointer dereferencing in C

```
movq (%rcx),%rax
```

- Displacement D(R) Mem[Reg[R]+D]
  - Register R specifies start of memory region
  - Constant displacement D specifies offset (which can be positive or negative)

```
movq 8(%rbp),%rdx
```

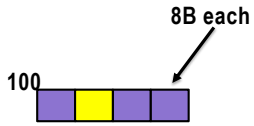
21

21

movq 8(%rbp), %rdx

Assume %rbp holds 100

```
long arr_long[4];
long tmp = arr_long[1];
```



22

22

Complete Memory Addressing Modes

- Most General Form

  D(Rb,Ri,S) Mem[Reg[Rb]+S*Reg[Ri]+ D]
  - D: Constant "displacement" stored in 1, 2, or 4 bytes
  - Rb: Base register: Any of 16 integer registers
  - Ri: Index register: Any, except for `%rsp`
  - S: Scale: 1, 2, 4, or 8

- Special Cases

  (Rb,Ri) Mem[Reg[Rb]+Reg[Ri]]

  D(Rb,Ri) Mem[Reg[Rb]+Reg[Ri]+D]

  (Rb,Ri,S) Mem[Reg[Rb]+S*Reg[Ri]]

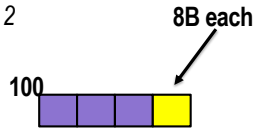
23

23

movq 8(%rbp, %rdi, 8), %rdx

Assume %rbp holds 100, %rdi holds 2

```
long arr_long[4];
long tmp = arr_long[3];
```



24

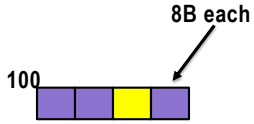
24

Practice On Your Own

Assume %rbp holds 100

```
long arr_long[4];

// Write code in assembly
arr_long[2] = 7;
```



25

25

Address Computation Examples

`%rdx`	`0xf000`
`%rcx`	`0x0100`

Most General Form

D(Rb,Ri,S) **Mem[Reg[Rb]+S*Reg[Ri]+ D]**

- D: Constant "displacement" 1, 2, or 4 bytes
- Rb: Base register: Any of 16 integer registers
- Ri: Index register: Any, except for %rsp
- S: Scale: 1, 2, 4, or 8

Expression	Address Computation	Address
`0x8(%rdx)`	`0xf000 + 0x8`	`0xf008`
`(%rdx,%rcx)`	`0xf000 + 0x100`	`0xf100`
`(%rdx,%rcx,4)`	`0xf000 + 4*0x100`	`0xf400`
`0x80(,%rdx,2)`	`2*0xf000 + 0x80`	`0x1e080`

26

26