System Architecture and Assembly

Systems Programming
(CST-210)

A. P. MAZUMDAR

Intel x86 Processors

- Totally dominate laptop/desktop/server market
- Evolutionary design
  - Backwards compatible up until 8086, introduced in 1978
  - Added more features as time goes on

Intel x86 Processors

- Complex instruction set computer (CISC)
  - Many different instructions with many different formats
    - But, only small subset encountered with Linux programs
  - Hard to match performance of Reduced Instruction Set Computers (RISC)
  - But, Intel has done just that!
    - In terms of speed. Less so for low power.

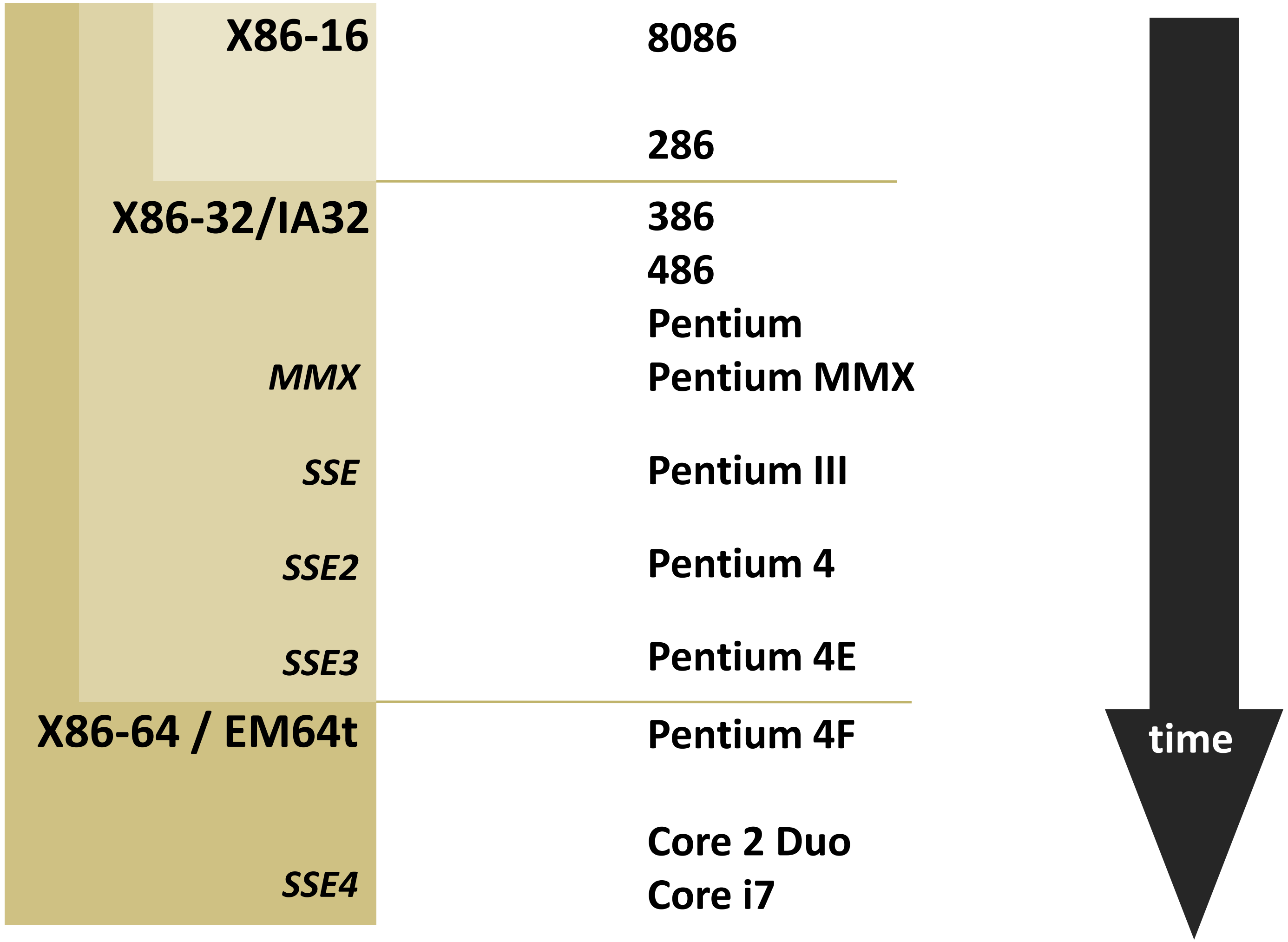
Intel x86 Evolution: Milestones

Name *Date* *Transistors* *MHz*

- **8086** **1978** **29K** **5-10**
  - First 16-bit processor. Basis for IBM PC & DOS
  - 1MB address space
- **386** **1985** **275K** **16-33**
  - First 32 bit processor , referred to as IA32
  - Added “flat addressing”
  - Capable of running Unix
  - 32-bit Linux/gcc uses no instructions introduced in later models

Intel x86 Evolution: Milestones

Name	*Date*	*Transistors*	*MHz*
Pentium 4F	**2004**	**125M**	**2800-3800**
First 64-bit processor, referred to as x86-64			
Core i7	**2008**	**731M**	**2667-3333**
New machines			



Architectures	Processors
X86-16	8086
	286
X86-32/IA32	386
	486
	Pentium
MMX	Pentium MMX
SSE	Pentium III
SSE2	Pentium 4
SSE3	Pentium 4E
X86-64 / EM64t	Pentium 4F
	Core 2 Duo
SSE4	Core i7

IA: often redefined as latest Intel architecture

x86 Clones: Advanced Micro Devices (AMD)

- **Historically**
  - AMD has followed just behind Intel
  - A little bit slower, a lot cheaper
- **Then**
  - Recruited top circuit designers from Digital Equipment Corp. and other downward trending companies
  - Built Opteron: tough competitor to Pentium 4
  - Developed x86-64, their own extension to 64 bits

Intel’s 64-Bit

- Intel Attempted Radical Shift from IA32 to IA64
  - Totally different architecture (Itanium)
  - Executes IA32 code only as legacy
  - Performance disappointing
- AMD Stepped in with Evolutionary Solution
  - x86-64 (now called “AMD64”)
- Intel Felt Obligated to Focus on IA64
  - Hard to admit mistake or that AMD is better

Intel's 64-Bit

- 2004: Intel Announces EM64T extension to IA32
  - Extended Memory 64-bit Technology
  - Almost identical to x86-64!
- All but low-end x86 processors support x86-64
  - But, lots of code still runs in 32-bit mode

IA32 (Pentium) Processor Architecture

- 32 bit Processor
- 1 WORD = 16bit
- 32 bits = 2 WORDS = 1 Double Word (DWORD)

Processor modes

1. Protected (*important*)
   - 32-bit mode
   - 32-bit (4GB) address space
2. Virtual 8086 modes
3. Real mode
   - 1MB address space

20 bit address space, each pointing to 1Byte of memory
= 1 MebiBytes (MiB)

4. System management mode

Registers

- 32-bit GPR’s (“general” purpose registers):

eax

ebx

ecx

edx

eflags

ebp

esp

esi

edi

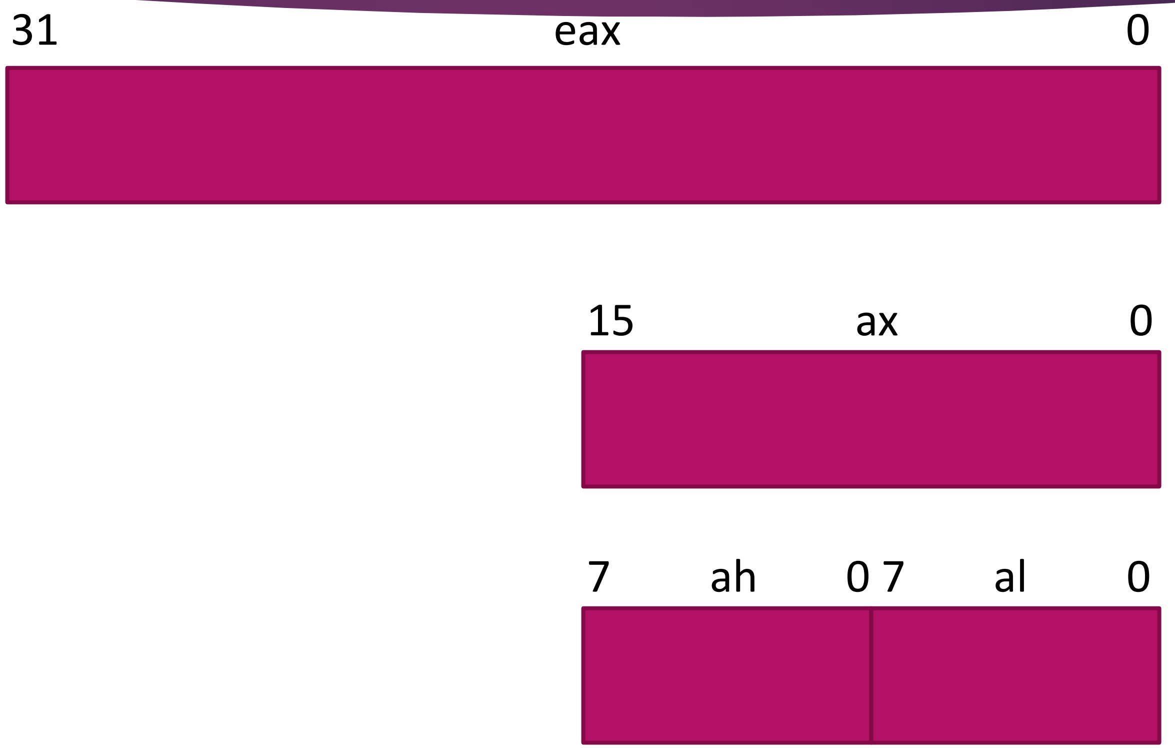
eip

e[a,b,c,d]x:

32 bits

16 bits

8 bits



Note: eax is **one** register that can be viewed **four** different ways.

Registers

- Not really GPR's.
  - eax - accumulator; multiplication and division
  - ecx - loop counter
  - esp - stack pointers; don't use
  - esi, edi - for memory-to-memory transfer
  - ebp - used by HLL for local vars on stack

Base Pointer

Registers

- Additional registers:
  - 16-bit segment registers
    - cs, es, ss, fs, ds, gs
    - don’t use
  - eip
    - instruction pointer / program counter (PC)
    - don‘t use

Registers

- CS
  - Address of current code segment
- SS
  - Address of current stack segment
- Others (DS, ES, FS, GS)
  - Address of data segments

Registers

- Additional registers:
  - eflags
    - contains results of operations
    - 32 individual bits
      - control flags
      - status flags:
        - **C = carry (unsigned)**
        - **O = overflow (signed); also called V**
        - **S = sign; also called N for negative**
        - **Z = zero**

Registers

- Additional registers:
  - floating point registers:
    - ST(0) … ST(7)
      - 80 bits
  - MMX has 8 64-bit regs No official meaning, For SIMD, Only for Integers
    - Translate segment address to Physical address
  - XMM has 8 128-bit regs for Streaming SIMD Extensions (SSE); Floats and Integers

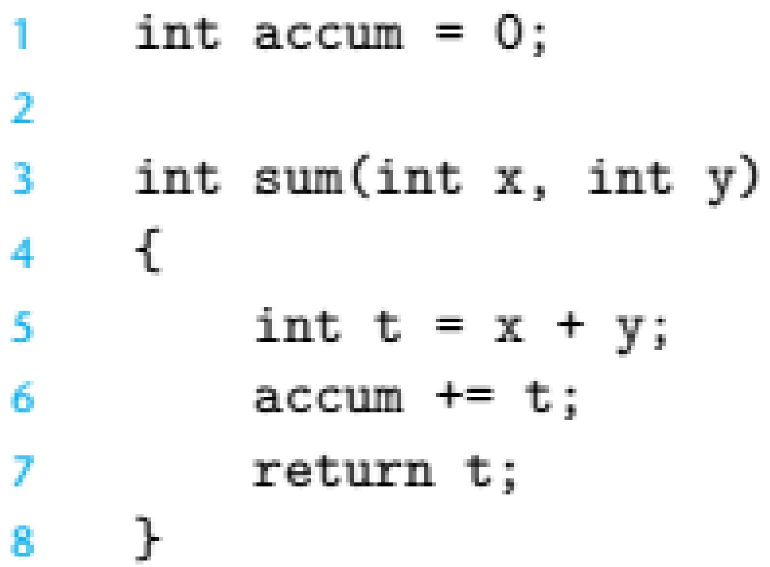
Compilation

- Two parts of a program: **p1.c and p2.c**
- To compile:
  - gcc -O1 -o p p1.c p2.c
- -o
  - Output file name, followed by the <name>
- -O1
  - Level of optimization: (higher level = faster execution, slower compilation)

Compilation

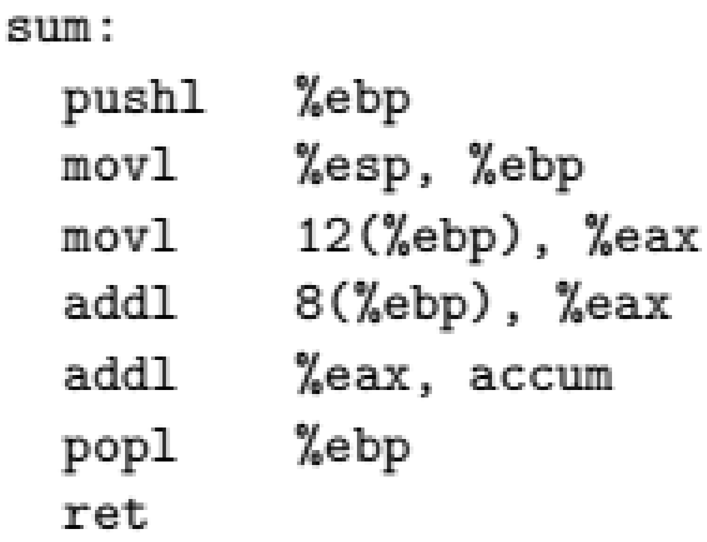
- ASM code generation:
  - ***gcc   -O1   -S   code.c***

- Output file   code.s will be generated
- -S :   gcc option to compile till   assembly level



```
int accum = 0;

int sum(int x, int y)
{
    int t = x + y;
    accum += t;
    return t;
}
```

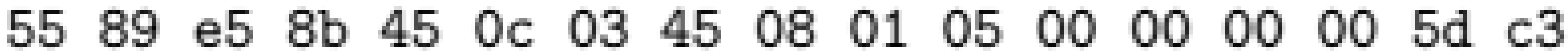


```
sum:
  pushl   %ebp
  movl    %esp, %ebp
  movl    12(%ebp), %eax
  addl    8(%ebp), %eax
  addl    %eax, accum
  popl    %ebp
  ret
```

Creating Object code

- **gcc –O1 -c code.c**
  - **-c:** option for generating obj code

- For code.c :



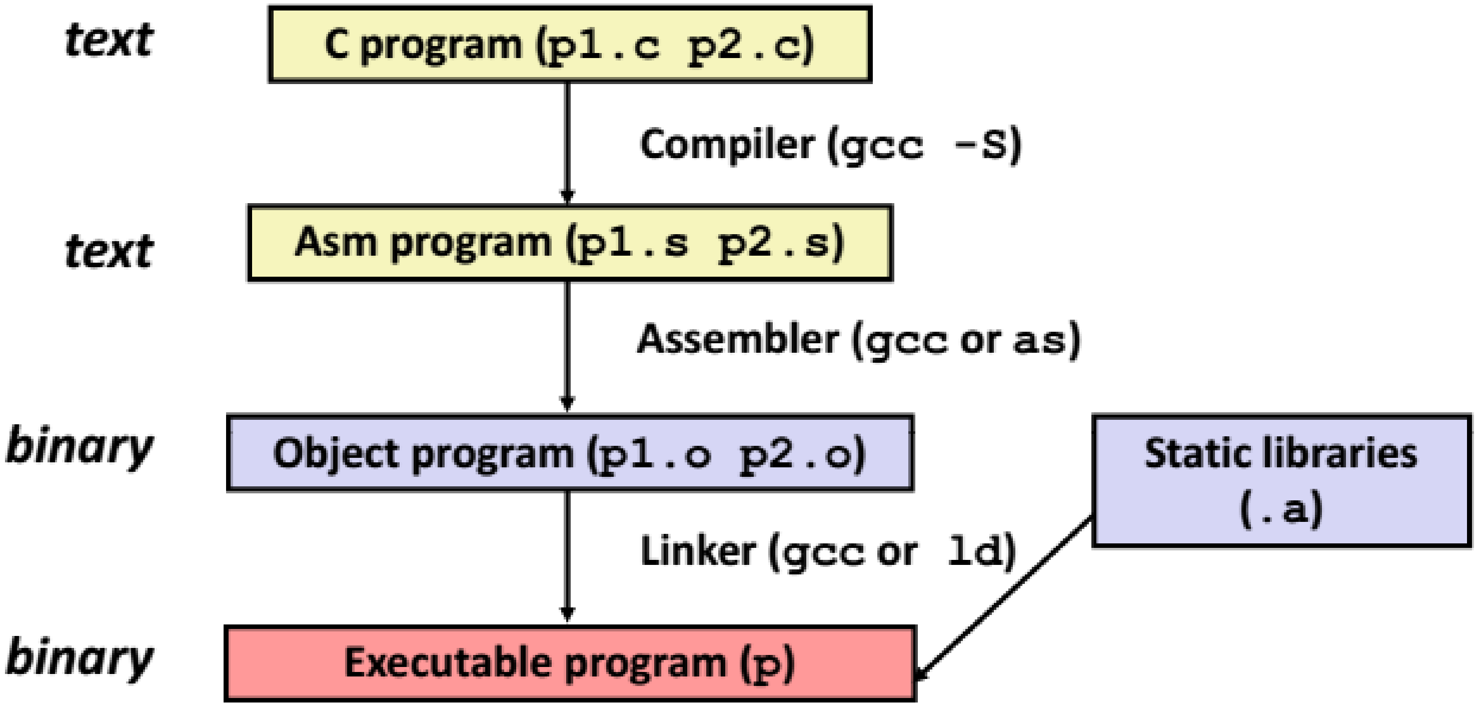
```
55 89 e5 8b 45 0c 03 45 08 01 05 00 00 00 00 5d c3
```

Generating Object code

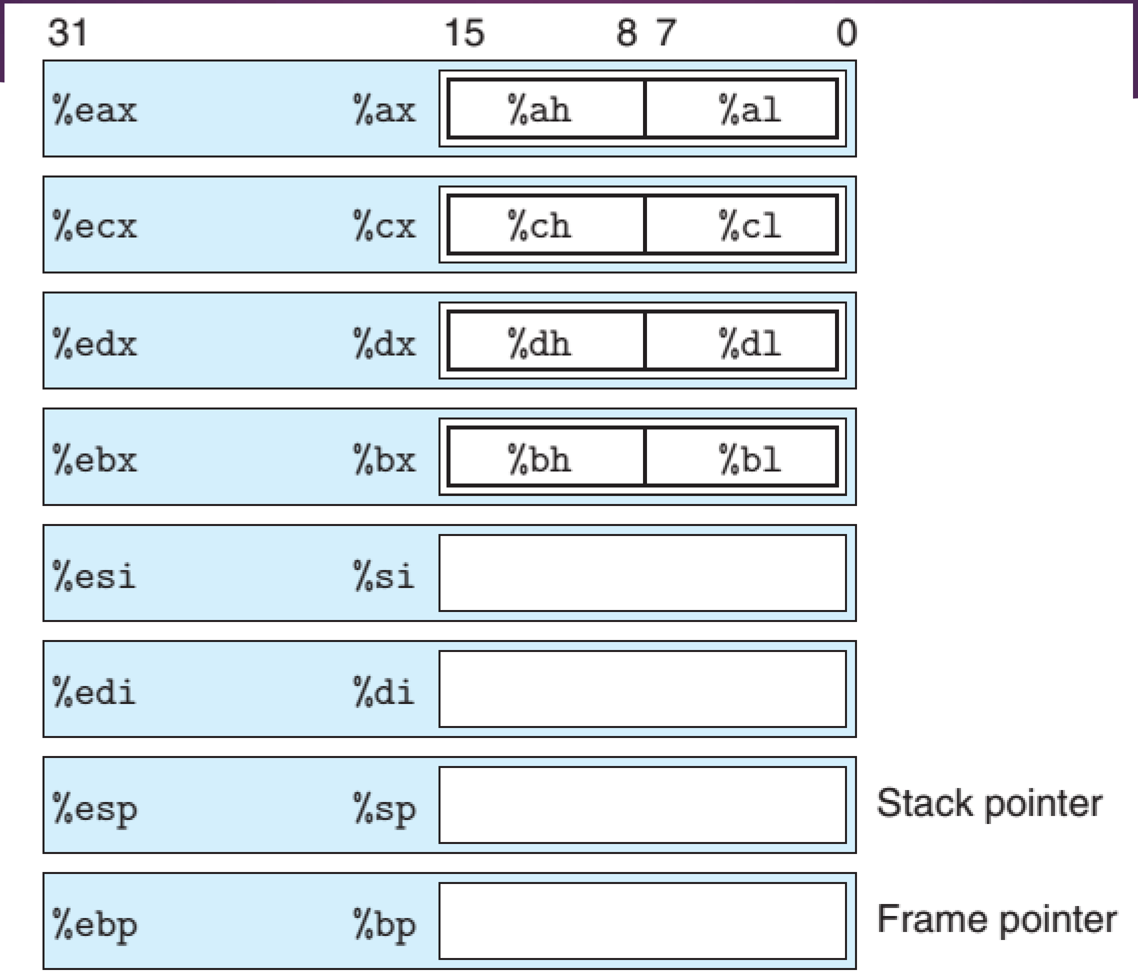
- **gcc –O1 -c code.c**
- Use Disassembler to byte code length
  - In this case it is 17

- Use GNU debugging toll (GDB) on **code.o** to get the code
  - **(gdb) x/17xb sum**

Summary of compilation

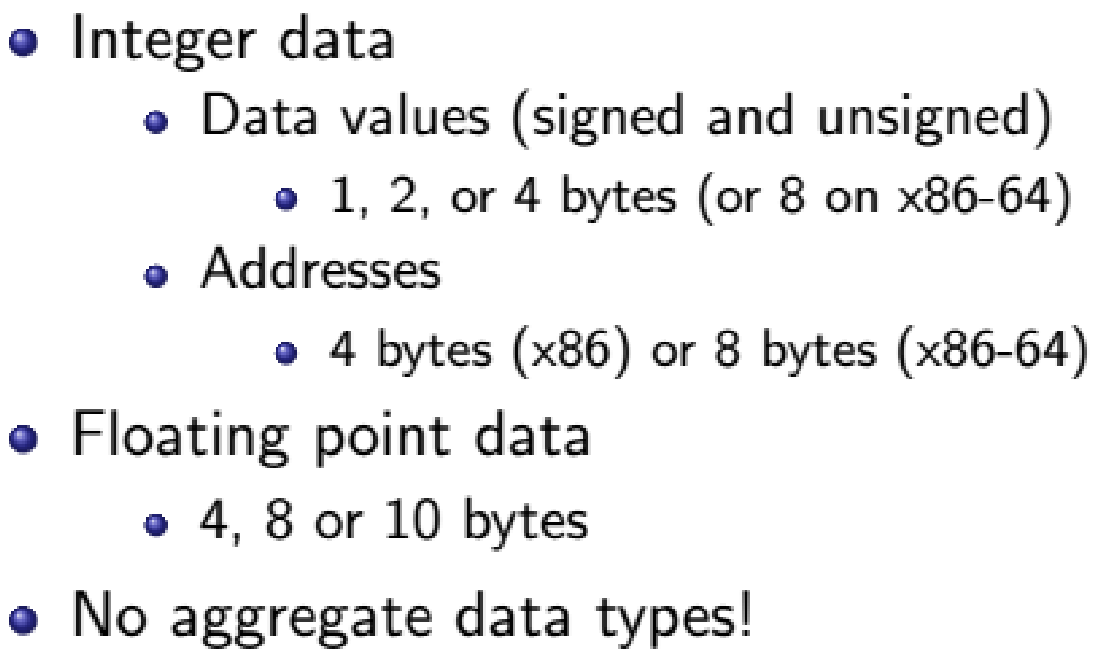


Revisit IA32 Integer Registers



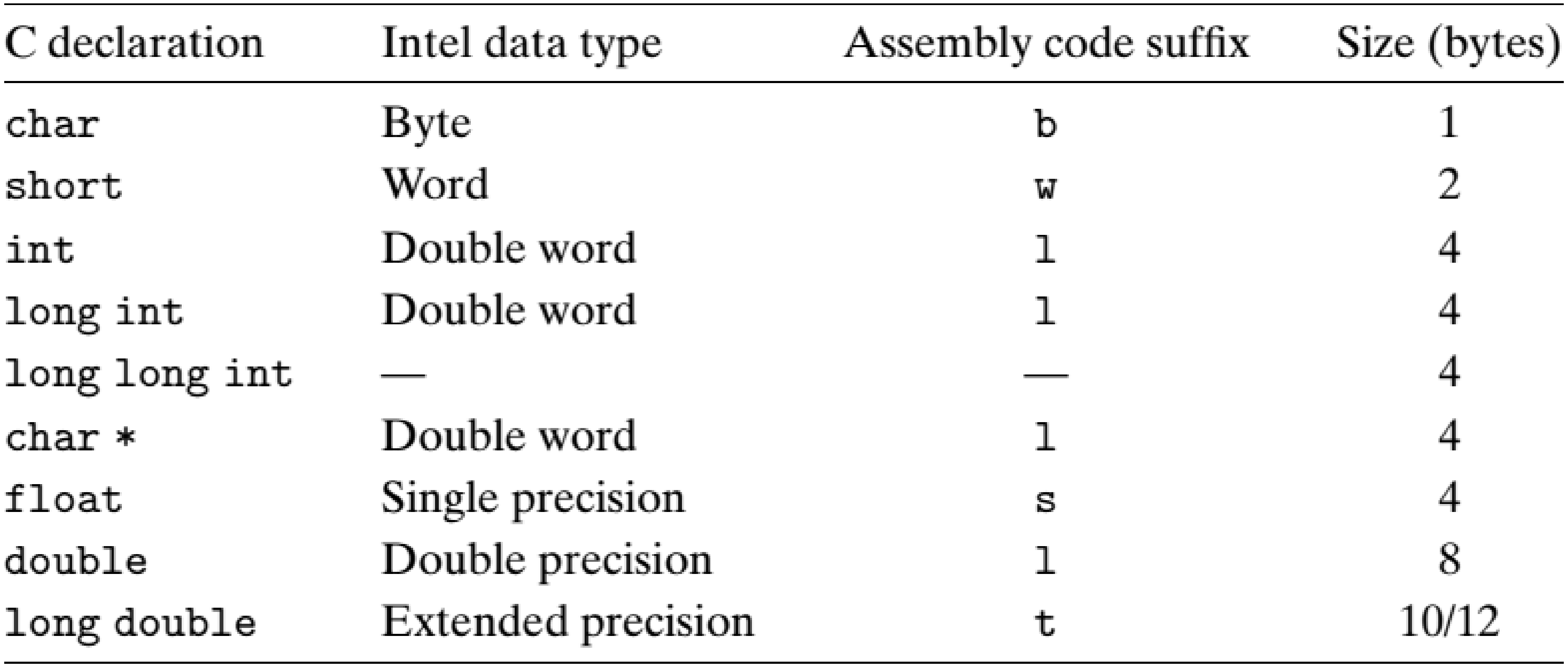
31	15	8 7	0	
%eax	%ax	%ah	%al	
%ecx	%cx	%ch	%cl	
%edx	%dx	%dh	%dl	
%ebx	%bx	%bh	%bl	
%esi	%si			
%edi	%di			
%esp	%sp			Stack pointer
%ebp	%bp			Frame pointer

Data types



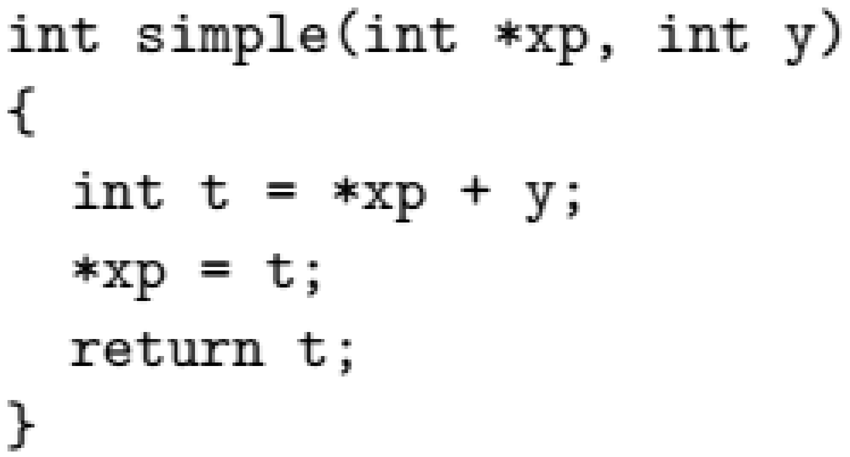
- Integer data
  - Data values (signed and unsigned)
    - 1, 2, or 4 bytes (or 8 on x86-64)
  - Addresses
    - 4 bytes (x86) or 8 bytes (x86-64)
- Floating point data
  - 4, 8 or 10 bytes
- No aggregate data types!

C Data Types in IA32

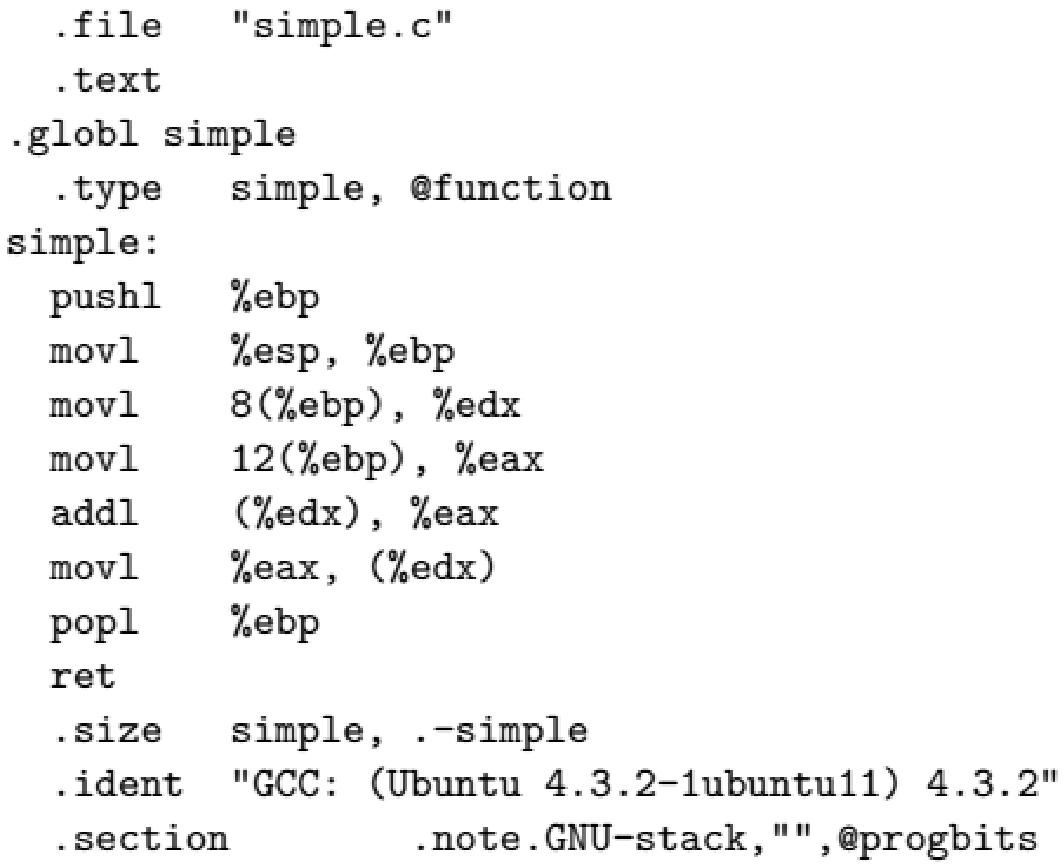


C declaration	Intel data type	Assembly code suffix	Size (bytes)
`char`	Byte	`b`	1
`short`	Word	`w`	2
`int`	Double word	`l`	4
`long int`	Double word	`l`	4
`long long int`	—	—	4
`char *`	Double word	`l`	4
`float`	Single precision	`s`	4
`double`	Double precision	`l`	8
`long double`	Extended precision	`t`	10/12

Assembly Conversion

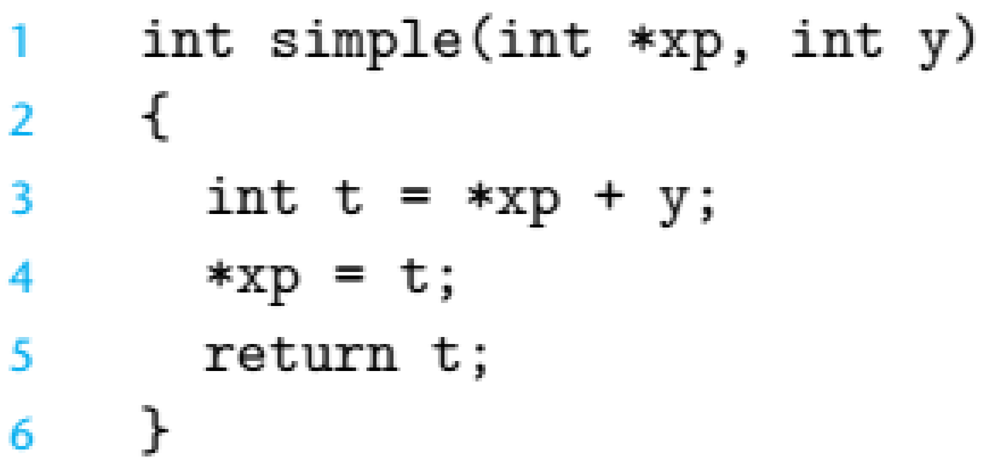


```
int simple(int *xp, int y)
{
  int t = *xp + y;
  *xp = t;
  return t;
}
```

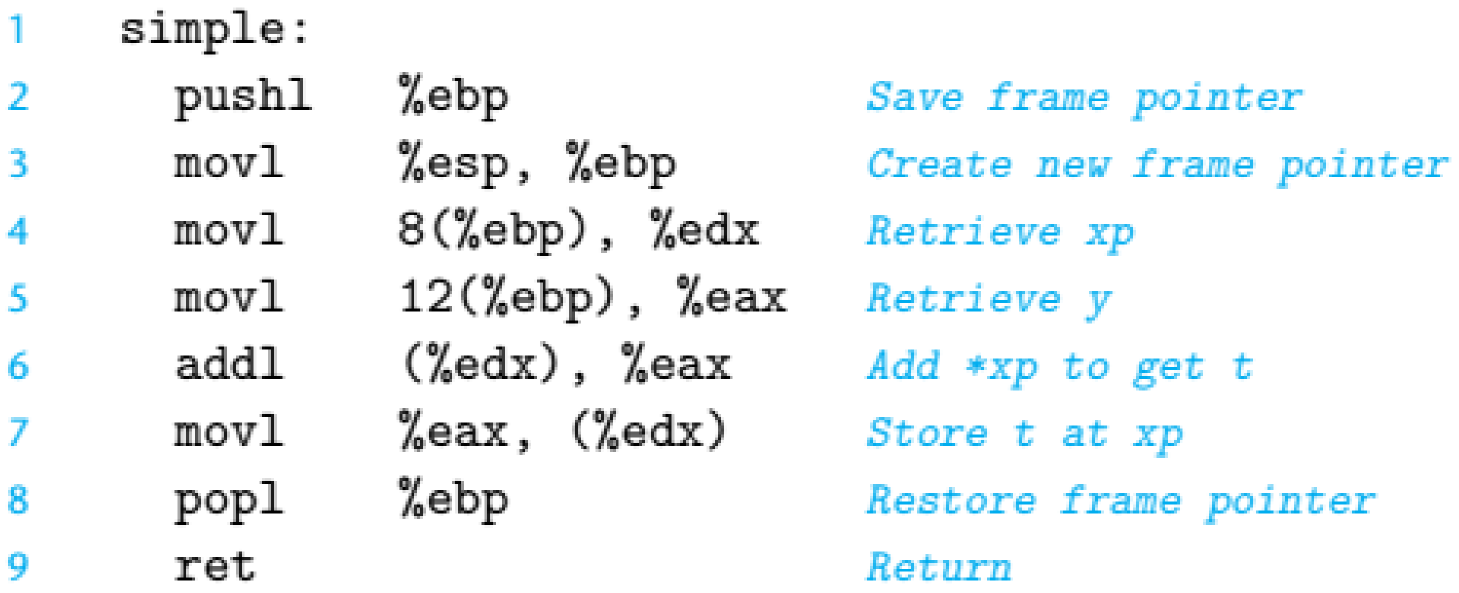


```
  .file   "simple.c"
  .text
.globl simple
  .type   simple, @function
simple:
  pushl   %ebp
  movl    %esp, %ebp
  movl    8(%ebp), %edx
  movl    12(%ebp), %eax
  addl    (%edx), %eax
  movl    %eax, (%edx)
  popl    %ebp
  ret
  .size   simple, .-simple
  .ident  "GCC: (Ubuntu 4.3.2-1ubuntu11) 4.3.2"
  .section        .note.GNU-stack,"",@progbits
```

Assembly Conversion



```
int simple(int *xp, int y)
{
  int t = *xp + y;
  *xp = t;
  return t;
}
```



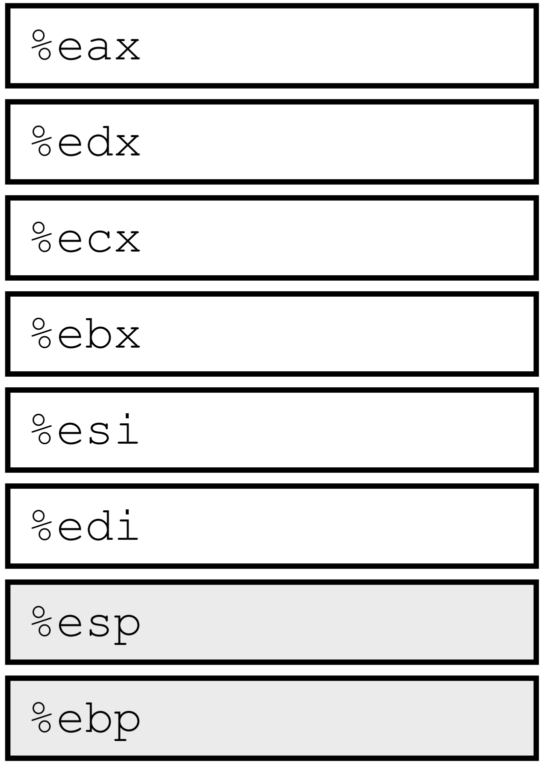
```
simple:
  pushl   %ebp            Save frame pointer
  movl    %esp, %ebp      Create new frame pointer
  movl    8(%ebp), %edx   Retrieve xp
  movl    12(%ebp), %eax  Retrieve y
  addl    (%edx), %eax    Add *xp to get t
  movl    %eax, (%edx)    Store t at xp
  popl    %ebp            Restore frame pointer
  ret                     Return
```

Move

- Moving Data

  `movl` ***Source*, *Dest*:**

  - Move 4-byte (“long”) word
  - Lots of these in typical code
- Operand Types
  - Immediate: Constant integer data
    - Like C constant, but prefixed with ‘`$`’
    - E.g., `$0x400`, `$-533`
    - Encoded with 1, 2, or 4 bytes



`%eax`

`%edx`

`%ecx`

`%ebx`

`%esi`

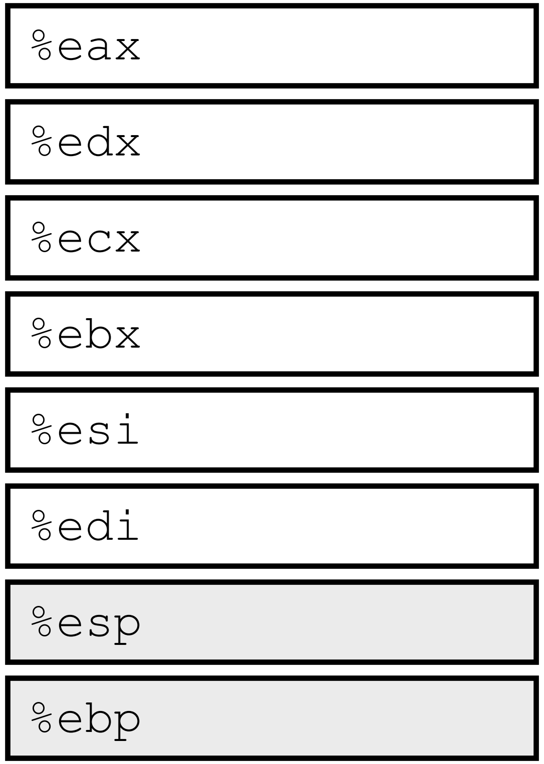
`%edi`

`%esp`

`%ebp`

Move

- Register: One of 8 integer registers
  - But `%esp` and `%ebp` reserved for special use
  - Others have special uses for particular instructions
- Memory: 4 consecutive bytes of memory
  - Various "address modes"



%eax

%edx

%ecx

%ebx

%esi

%edi

%esp

%ebp

Move

	Source	Destination		C Analog
`movl`	*Imm*	*Reg*	`movl $0x4,%eax`	`temp = 0x4;`
		Mem	`movl $-147,(%eax)`	`*p = -147;`
	Reg	*Reg*	`movl %eax,%edx`	`temp2 = temp1;`
		Mem	`movl %eax,(%edx)`	`*p = temp;`
	Mem	*Reg*	`movl (%eax),%edx`	`temp = *p;`

Simple Addressing Modes

- **Normal (R) Mem[Reg[R]]**
  - Register R specifies memory address

```
movl (%ecx),%eax
```

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

```
movl 8(%ebp),%edx
```

Example: *Simple Addressing Modes*

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

```
swap:
    pushl %ebp
    movl  %esp,%ebp
    pushl %ebx
```

Set Up

```
    movl  12(%ebp),%ecx
    movl  8(%ebp),%edx
    movl  (%ecx),%eax
    movl  (%edx),%ebx
    movl  %eax,(%edx)
    movl  %ebx,(%ecx)
```

Body

```
    movl  -4(%ebp),%ebx
    movl  %ebp,%esp
    popl  %ebp
    ret
```

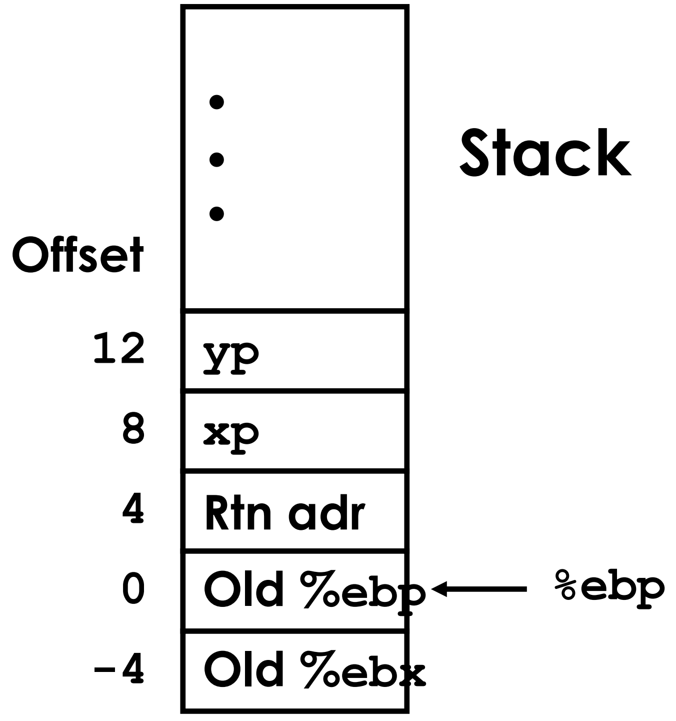
Finish

Swap Operation

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

Register	Variable
%ecx	yp
%edx	xp
%eax	t1
%ebx	t0

```
movl 12(%ebp),%ecx   # ecx = yp
movl 8(%ebp),%edx    # edx = xp
movl (%ecx),%eax     # eax = *yp (t1)
movl (%edx),%ebx     # ebx = *xp (t0)
movl %eax,(%edx)     # *xp = eax
movl %ebx,(%ecx)     # *yp = ebx
```



Swap Operation

%eax	
%edx	
%ecx	
%ebx	
%esi	
%edi	
%esp	
%ebp	0x104

```
movl 12(%ebp),%ecx  # ecx = yp
movl 8(%ebp),%edx   # edx = xp
movl (%ecx),%eax    # eax = *yp (t1)
movl (%edx),%ebx    # ebx = *xp (t0)
movl %eax,(%edx)    # *xp = eax
movl %ebx,(%ecx)    # *yp = ebx
```

	Offset		Address
		123	0x124
		456	0x120
			0x11c
			0x118
			0x114
yp	12	0x120	0x110
xp	8	0x124	0x10c
	4	Rtn adr	0x108
%ebp →	0		0x104
	-4		0x100

Swap Operation

%eax	
%edx	
%ecx	0x120
%ebx	
%esi	
%edi	
%esp	
%ebp	0x104

```
movl 12(%ebp),%ecx   # ecx = yp
movl 8(%ebp),%edx    # edx = xp
movl (%ecx),%eax     # eax = *yp (t1)
movl (%edx),%ebx     # ebx = *xp (t0)
movl %eax,(%edx)     # *xp = eax
movl %ebx,(%ecx)     # *yp = ebx
```

	Offset		Address
		123	0x124
		456	0x120
			0x11c
			0x118
			0x114
yp	12	0x120	0x110
xp	8	0x124	0x10c
	4	Rtn adr	0x108
%ebp →	0		0x104
	-4		0x100

Swap Operation

%eax	
%edx	0x124
%ecx	0x120
%ebx	
%esi	
%edi	
%esp	
%ebp	0x104

```
movl 12(%ebp),%ecx   # ecx = yp
movl 8(%ebp),%edx    # edx = xp
movl (%ecx),%eax     # eax = *yp (t1)
movl (%edx),%ebx     # ebx = *xp (t0)
movl %eax,(%edx)     # *xp = eax
movl %ebx,(%ecx)     # *yp = ebx
```

	Offset		Address
		123	0x124
		456	0x120
			0x11c
			0x118
			0x114
yp	12	0x120	0x110
xp	8	0x124	0x10c
	4	Rtn adr	0x108
%ebp →	0		0x104
	-4		0x100

Swap Operation

%eax	456
%edx	0x124
%ecx	0x120
%ebx	
%esi	
%edi	
%esp	
%ebp	0x104

```
movl 12(%ebp),%ecx  # ecx = yp
movl 8(%ebp),%edx   # edx = xp
movl (%ecx),%eax    # eax = *yp (t1)
movl (%edx),%ebx    # ebx = *xp (t0)
movl %eax,(%edx)    # *xp = eax
movl %ebx,(%ecx)    # *yp = ebx
```

	Offset		Address
		123	0x124
		456	0x120
			0x11c
			0x118
			0x114
yp	12	0x120	0x110
xp	8	0x124	0x10c
	4	Rtn adr	0x108
%ebp →	0		0x104
	-4		0x100

Swap Operation

%eax	456
%edx	0x124
%ecx	0x120
%ebx	123
%esi	
%edi	
%esp	
%ebp	0x104

```
movl 12(%ebp),%ecx  # ecx = yp
movl 8(%ebp),%edx   # edx = xp
movl (%ecx),%eax    # eax = *yp (t1)
movl (%edx),%ebx    # ebx = *xp (t0)
movl %eax,(%edx)    # *xp = eax
movl %ebx,(%ecx)    # *yp = ebx
```

	Offset		Address
		123	0x124
		456	0x120
			0x11c
			0x118
			0x114
yp	12	0x120	0x110
xp	8	0x124	0x10c
	4	Rtn adr	0x108
%ebp →	0		0x104
	-4		0x100

Swap Operation

%eax	456
%edx	0x124
%ecx	0x120
%ebx	123
%esi	
%edi	
%esp	
%ebp	0x104

```
movl 12(%ebp),%ecx   # ecx = yp
movl 8(%ebp),%edx    # edx = xp
movl (%ecx),%eax     # eax = *yp (t1)
movl (%edx),%ebx     # ebx = *xp (t0)
movl %eax,(%edx)     # *xp = eax
movl %ebx,(%ecx)     # *yp = ebx
```

	Offset		Address
		456	0x124
		456	0x120
			0x11c
			0x118
			0x114
yp	12	0x120	0x110
xp	8	0x124	0x10c
	4	Rtn adr	0x108
%ebp →	0		0x104
	-4		0x100

Swap Operation

%eax	456
%edx	0x124
%ecx	0x120
%ebx	123
%esi	
%edi	
%esp	
%ebp	0x104

```
movl 12(%ebp),%ecx   # ecx = yp
movl 8(%ebp),%edx    # edx = xp
movl (%ecx),%eax     # eax = *yp (t1)
movl (%edx),%ebx     # ebx = *xp (t0)
movl %eax,(%edx)     # *xp = eax
movl %ebx,(%ecx)     # *yp = ebx
```

	Offset		Address
		456	0x124
		123	0x120
			0x11c
			0x118
			0x114
yp	12	0x120	0x110
xp	8	0x124	0x10c
	4	Rtn adr	0x108
%ebp →	0		0x104
	-4		0x100

Arithmetic Operations

- Two-Operand Instructions

Format	Computation	
`addl` *Src,Dest*	*Dest = Dest + Src*	
`subl` *Src,Dest*	*Dest = Dest − Src*	
`imull` *Src,Dest*	*Dest = Dest * Src*	
`sall` *k,Dest*	*Dest = Dest << k*	**Also called `shll`**
`sarl` *k,Dest*	*Dest = Dest >> k*	**Arithmetic**
`shrl` *k,Dest*	*Dest = Dest >> k*	**Logical**

k is an immediate value or contents of %cl

Arithmetic Operations

Format	Computation
- Two-Operand Instructions	
`xorl` ***Src,Dest***	***Dest*** = ***Dest*** ^ ***Src***
`andl` ***Src,Dest***	***Dest*** = ***Dest*** & ***Src***
`orl` ***Src,Dest***	***Dest*** = ***Dest*** | ***Src***

Arithmetic Operations

Format	Computation
One-Operand Instructions	
`incl` ***Dest***	***Dest*** = ***Dest*** + 1
`decl` ***Dest***	***Dest*** = ***Dest*** - 1
`negl` ***Dest***	***Dest*** = -***Dest***
`notl` ***Dest***	***Dest*** = ~***Dest***

Assembler Directives

```
int main() {
    int a,b;
    a = 10;
    b=a+5;

    return b;
}
```

```
    .file    "t1.c"
    .text
    .globl    main
    .type    main, @function
main:
.LFB0:
    .cfi_startproc
    pushl    %ebp
    .cfi_def_cfa_offset 8
    .cfi_offset 5, -8
    movl    %esp, %ebp
    .cfi_def_cfa_register 5
```

```
    subl    $16, %esp
    movl    $10, -8(%ebp)
    movl    -8(%ebp), %eax
    addl    $5, %eax
    movl    %eax, -4(%ebp)
    movl    -4(%ebp), %eax
    leave
    .cfi_restore 5
    .cfi_def_cfa 4, 4
    ret
    .cfi_endproc

.LFE0:
    .size    main, .-main
    .ident    "GCC: (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3"
    .section    .note.GNU-stack,"",@progbits
```

Indexed Addressing Modes

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 8 integer registers
  - Ri: Index register: Any, **except for `%esp`**
    - Unlikely you'd use `%ebp`, either
  - S: Scale: 1, 2, 4, or 8

Indexed Addressing Modes

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]]**

Example

%edx	0xf000
%ecx	0x100

Expression
0x8(%edx)
(%edx,%ecx)
(%edx,%ecx,4)
0x80(,%edx,2)

Address Computation Instruction

- `leal` ***Src*,*Dest***

  Load Effective Address

  - *Src* is address mode expression
  - Set *Dest* to address denoted by expression

- Uses
  - Computing address without doing memory reference
    - E.g., translation of `p = &x[i];`
  - Computing arithmetic expressions of the form x + k*y

    x + k*y + z
    - k = 1, 2, 4, or 8.

leal Example

```
int arith
  (int x, int y, int z)
{
  int t1 = x+y;
  int t2 = z+t1;
  int t3 = x+4;
  int t4 = y * 48;
  int t5 = t3 + t4;
  int rval = t2 * t5;
  return rval;
}
```

```
arith:
    pushl %ebp
    movl %esp,%ebp

    movl 8(%ebp),%eax
    movl 12(%ebp),%edx
    leal (%edx,%eax),%ecx
    leal (%edx,%edx,2),%edx
    sall $4,%edx
    addl 16(%ebp),%ecx
    leal 4(%edx,%eax),%eax
    imull %ecx,%eax

    movl %ebp,%esp
    popl %ebp
    ret
```

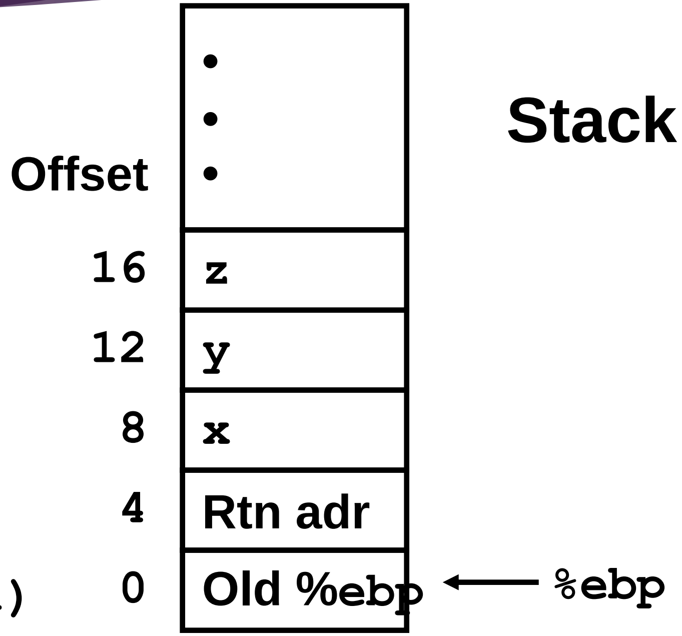
Set Up

Body

Finish

leal Example

```
int arith
   (int x, int y, int z)
{
  int t1 = x+y;
  int t2 = z+t1;
  int t3 = x+4;
  int t4 = y * 48;
  int t5 = t3 + t4;
  int rval = t2 * t5;
  return rval;
}
```



```
movl  8(%ebp),%eax         # eax = x
movl  12(%ebp),%edx        # edx = y
leal  (%edx,%eax),%ecx     # ecx = x+y   (t1)
leal  (%edx,%edx,2),%edx   # edx = 3*y
sall  $4,%edx              # edx = 48*y (t4)
addl  16(%ebp),%ecx        # ecx = z+t1  (t2)
leal  4(%edx,%eax),%eax    # eax = 4+t4+x  (t5)
imull  %ecx,%eax           # eax = t5*t2  (rval)
```

leal Example

```
int arith
  (int x, int y, int z)
{
  int t1 = x+y;
  int t2 = z+t1;
  int t3 = x+4;
  int t4 = y * 48;
  int t5 = t3 + t4;
  int rval = t2 * t5;
  return rval;
}
```

```
# eax = x
  movl 8(%ebp),%eax
# edx = y
  movl 12(%ebp),%edx
# ecx = x+y  (t1)
  leal (%edx,%eax),%ecx
# edx = 3*y
  leal (%edx,%edx,2),%edx
# edx = 48*y (t4)
  sall $4,%edx
# ecx = z+t1 (t2)
  addl 16(%ebp),%ecx
# eax = 4+t4+x (t5)
  leal 4(%edx,%eax),%eax
# eax = t5*t2 (rval)
  imull %ecx,%eax
```

48 = 3 * 16

4 <<

```
.file	"t2.c"
	.section	.rodata
.LC0:
	.string	"Result is %d\n"
	.text
	.globl	main
	.type	main, @function
main:
.LFB0:
	.cfi_startproc
	pushl	%ebp
	.cfi_def_cfa_offset 8
	.cfi_offset 5, -8
	movl	%esp, %ebp
	.cfi_def_cfa_register 5
	andl	$-16, %esp
	subl	$32, %esp
	movl	$10, 20(%esp)
	movl	$0, 24(%esp)
	movl	$4, 28(%esp)

	movl	$.LC0, %eax
	movl	28(%esp), %edx

	movl	%edx, 4(%esp)
	movl	%eax, (%esp)
	call	printf
	movl	$0, %eax
	leave
	.cfi_restore 5
	.cfi_def_cfa 4, 4
	ret
	.cfi_endproc
.LFE0:
	.size	main, .-main
	.ident	"GCC: (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3"
	.section	.note.GNU-stack,"",@progbits
 .file    "t2.c"
    .section    .rodata
.LC0:
    .string    "Result is %d\n"
    .text
    .globl    main
    .type    main, @function
main:
.LFB0:
    .cfi_startproc
    pushl    %ebp
    .cfi_def_cfa_offset 8
    .cfi_offset 5, -8
    movl    %esp, %ebp
    .cfi_def_cfa_register 5
    andl    $-16, %esp
    subl    $32, %esp
```

```
movl    $10, 20(%esp)
movl    $0, 24(%esp)
movl    $4, 28(%esp)

movl    $.LC0, %eax
movl    28(%esp), %edx

movl    %edx, 4(%esp)
movl    %eax, (%esp)
call    printf
movl    $0, %eax
leave
```

```
 .cfi_restore 5
    .cfi_def_cfa 4, 4
    ret
    .cfi_endproc
.LFE0:
    .size    main, .-main
    .ident    "GCC: (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3"
    .section    .note.GNU-stack,"",@progbits
```

```
.file   "t2.c"
    .section    .rodata
.LC0:
    .string   "Result is %d\n"
    .text
    .globl   main
    .type   main, @function
main:
.LFB0:
    .cfi_startproc
    pushl   %ebp
    .cfi_def_cfa_offset 8
    .cfi_offset 5, -8
    movl    %esp, %ebp
    .cfi_def_cfa_register 5
    andl    $-16, %esp
    subl    $32, %esp
```

```
movl    $10, 20(%esp)
movl    $0, 24(%esp)
movl    $4, 28(%esp)

movl    $.LC0, %eax
movl    28(%esp), %edx

movl    %edx, 4(%esp)
movl    %eax, (%esp)
call    printf
movl    $0, %eax
leave
```

```
 .cfi_restore 5
    .cfi_def_cfa 4, 4
    ret
    .cfi_endproc
.LFE0:
    .size    main, .-main
    .ident    "GCC: (Ubuntu/Linaro 4.6.3-
1ubuntu5) 4.6.3"
    .section    .note.GNU-
stack,"",@progbits
```

```
.file	"t2.c"
	.section	.rodata
.LC0:
	.string	"Result is %d\n"
	.text
	.globl	main
	.type	main, @function
main:
.LFB0:
	.cfi_startproc
	pushl	%ebp
	.cfi_def_cfa_offset 8
	.cfi_offset 5, -8
	movl	%esp, %ebp
	.cfi_def_cfa_register 5
	andl	$-16, %esp
	subl	$32, %esp
```

```
movl	$10, 20(%esp)
movl	$0, 24(%esp)
movl	$4, 28(%esp)

movl	$.LC0, %eax
movl	28(%esp), %edx

movl	%edx, 4(%esp)
movl	%eax, (%esp)
call	printf
movl	$0, %eax
leave
```

```
 .cfi_restore 5
	.cfi_def_cfa 4, 4
	ret
	.cfi_endproc
.LFE0:
	.size	main, .-main
	.ident	"GCC: (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3"
	.section	.note.GNU-stack,"",@progbits
 .file	"t2.c"
	.section	.rodata
.LC0:
	.string	"Result is %d\n"
	.text
	.globl	main
	.type	main, @function
main:
.LFB0:
	.cfi_startproc
	pushl	%ebp
	.cfi_def_cfa_offset 8
	.cfi_offset 5, -8
	movl	%esp, %ebp
	.cfi_def_cfa_register 5
	andl	$-16, %esp
	subl	$32, %esp
	movl	$10, 20(%esp)
	movl	$0, 24(%esp)
	movl	$4, 28(%esp)

	movl	$.LC0, %eax
	movl	28(%esp), %edx

	movl	%edx, 4(%esp)
	movl	%eax, (%esp)
	call	printf
	movl	$0, %eax
	leave
 .cfi_restore 5
	.cfi_def_cfa 4, 4
	ret
	.cfi_endproc
.LFE0:
	.size	main, .-main
	.ident	"GCC: (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3"
	.section	.note.GNU-stack,"",@progbits
	.file	"t2.c"
	.section	.rodata
.LC0:
	.string	"Result is %d\n"
	.text
	.globl	main
	.type	main, @function
main:
.LFB0:
	.cfi_startproc
	pushl	%ebp
	.cfi_def_cfa_offset 8
	.cfi_offset 5, -8
	movl	%esp, %ebp
	.cfi_def_cfa_register 5
	andl	$-16, %esp
	subl	$32, %esp
```

```
movl	$10, 20(%esp)
movl	$0, 24(%esp)
movl	$4, 28(%esp)

movl	$.LC0, %eax
movl	28(%esp), %edx

movl	%edx, 4(%esp)
movl	%eax, (%esp)
call	printf
movl	$0, %eax
leave
```

```
	.cfi_restore 5
	.cfi_def_cfa 4, 4
	ret
	.cfi_endproc
.LFE0:
	.size	main, .-main
	.ident	"GCC: (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3"
	.section	.note.GNU-stack,"",@progbits
	.file	"t2.c"
	.section	.rodata
.LC0:
	.string	"Result is %d\n"
	.text
	.globl	main
	.type	main, @function
main:
.LFB0:
	.cfi_startproc
	pushl	%ebp
	.cfi_def_cfa_offset 8
	.cfi_offset 5, -8
	movl	%esp, %ebp
	.cfi_def_cfa_register 5
	andl	$-16, %esp
	subl	$32, %esp
	movl	$10, 20(%esp)
	movl	$0, 24(%esp)
	movl	$4, 28(%esp)

	movl	$.LC0, %eax
	movl	28(%esp), %edx

	movl	%eax, (%esp)
	movl	%edx, 4(%esp)
	call	printf
	movl	$0, %eax
	leave
	.cfi_restore 5
	.cfi_def_cfa 4, 4
	ret
	.cfi_endproc
.LFE0:
	.size	main, .-main
	.ident	"GCC: (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3"
	.section	.note.GNU-stack,"",@progbits
```

```
.file	"t2.c"
	.section	.rodata
.LC0:
	.string	"Result is %d\n"
	.text
	.globl	main
	.type	main, @function
main:
.LFB0:
	.cfi_startproc
	pushl	%ebp
	.cfi_def_cfa_offset 8
	.cfi_offset 5, -8
	movl	%esp, %ebp
	.cfi_def_cfa_register 5
	andl	$-16, %esp
	subl	$32, %esp
```

```
movl	$10, 20(%esp)
movl	$0, 24(%esp)
movl	$4, 28(%esp)

movl	$.LC0, %eax
movl	28(%esp), %edx

movl	%edx, 4(%esp)
movl	%eax, (%esp)
call	printf
movl	$0, %eax
leave
```

```
 .cfi_restore 5
	.cfi_def_cfa 4, 4
	ret
	.cfi_endproc
.LFE0:
	.size	main, .-main
	.ident	"GCC: (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3"
	.section	.note.GNU-stack,"",@progbits
```

```
.file	"t2.c"
	.section	.rodata
.LC0:
	.string	"Result is %d\n"
	.text
	.globl	main
	.type	main, @function
main:
.LFB0:
	.cfi_startproc
	pushl	%ebp
	.cfi_def_cfa_offset 8
	.cfi_offset 5, -8
	movl	%esp, %ebp
	.cfi_def_cfa_register 5
	andl	$-16, %esp
	subl	$32, %esp
	movl	$10, 20(%esp)
	movl	$0, 24(%esp)
	movl	$4, 28(%esp)

	movl	$.LC0, %eax
	movl	28(%esp), %edx

	movl	%edx, 4(%esp)
	movl	%eax, (%esp)
	call	printf
	movl	$0, %eax
	leave
	.cfi_restore 5
	.cfi_def_cfa 4, 4
	ret
	.cfi_endproc
.LFE0:
	.size	main, .-main
	.ident	"GCC: (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3"
	.section	.note.GNU-stack,"",@progbits
```

```
.file    "t2.c"
    .section    .rodata
.LC0:
    .string    "Result is %d\n"
    .text
    .globl    main
    .type    main, @function
main:
.LFB0:
    .cfi_startproc
    pushl    %ebp
    .cfi_def_cfa_offset 8
    .cfi_offset 5, -8
    movl    %esp, %ebp
    .cfi_def_cfa_register 5
    andl    $-16, %esp
    subl    $32, %esp
```

```
movl    $10, 20(%esp)
movl    $0, 24(%esp)
movl    $4, 28(%esp)
movl    28(%esp), %edx
addl    $15, %edx
movl    %edx, 28(%esp)

movl    $.LC0, %eax
movl    28(%esp), %edx

movl    %edx, 4(%esp)
movl    %eax, (%esp)
call    printf
movl    28(%esp), %eax
leave
```

```
 .cfi_restore 5
    .cfi_def_cfa 4, 4
    ret
    .cfi_endproc
.LFE0:
    .size    main, .-main
    .ident    "GCC: (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3"
    .section    .note.GNU-stack,"",@progbits
```

Displaying the Return Value

- `./a.out`
- `echo $?`

SARL - Arithm
SHRL - Logical

imull
mull

cltd
idivl
divl