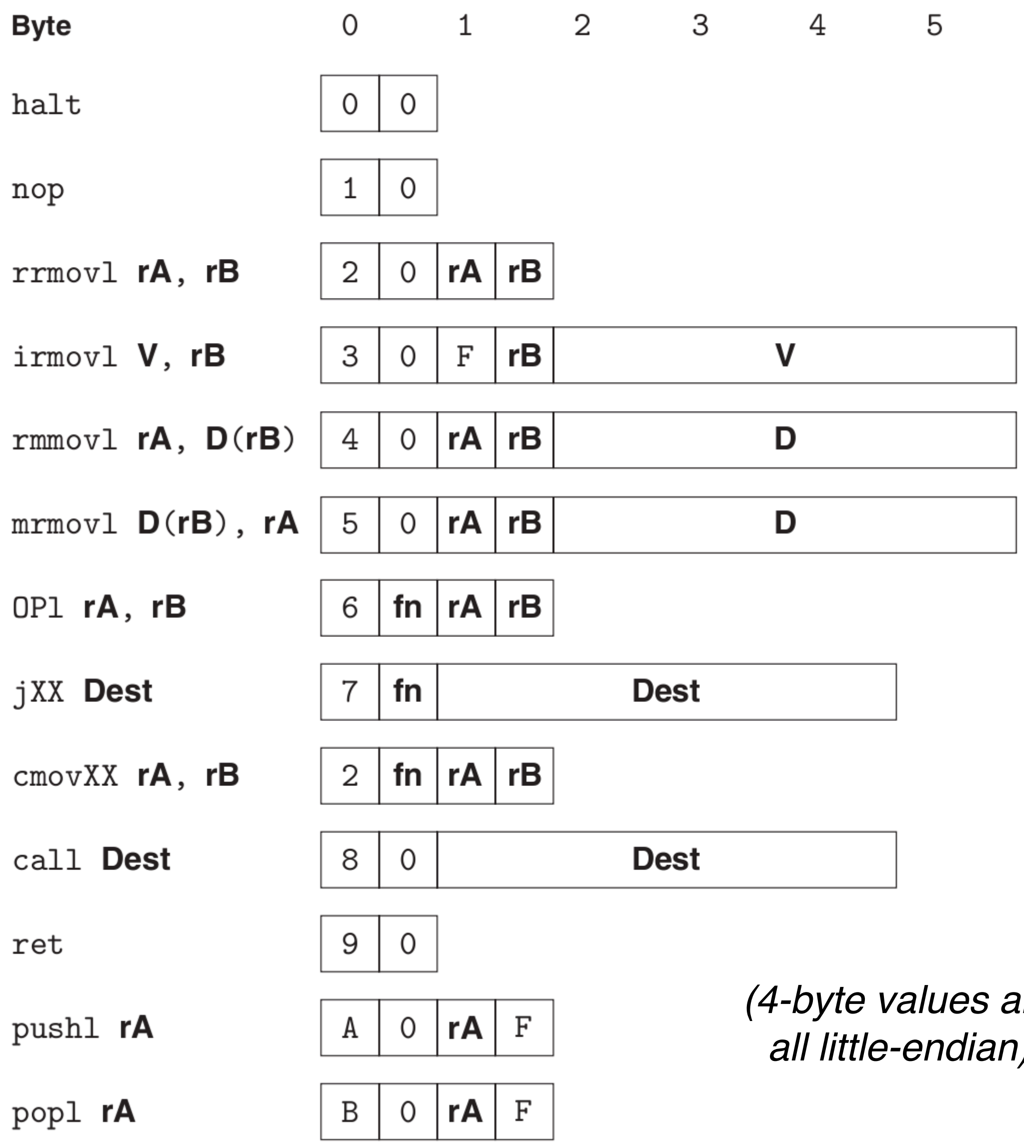
Re-Orienting

Value	Name	Meaning
1	AOK	Normal operation
2	HLT	halt instruction encountered
3	ADR	Invalid address encountered
4	INS	Invalid instruction encountered

Figure 4.5 **Y86 status codes.** In our design, the processor halts for any code other than AOK.

Number	Register name
0	%eax
1	%ecx
2	%edx
3	%ebx
4	%esp
5	%ebp
6	%esi
7	%edi
F	No register



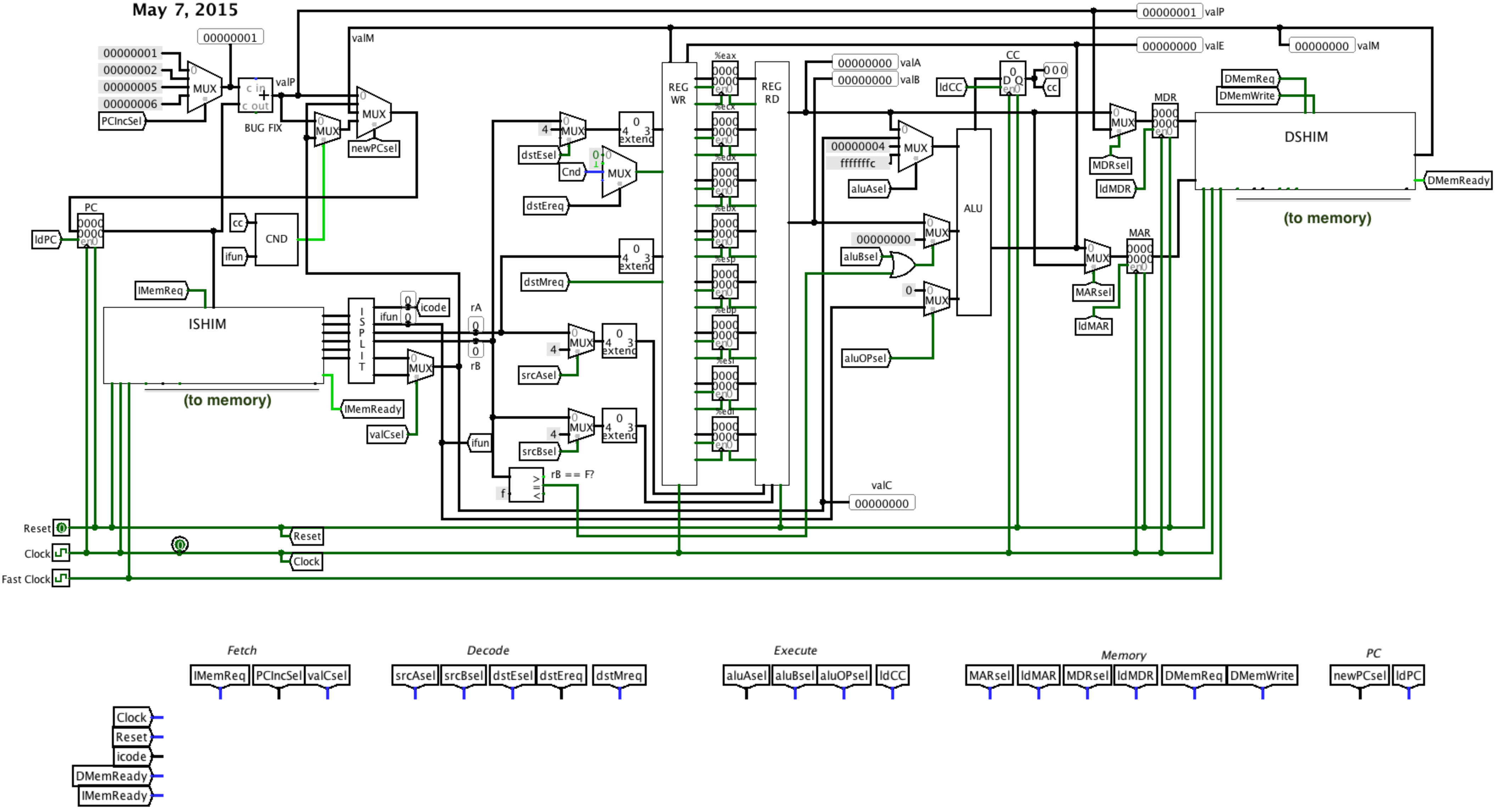
Byte	0		1		2	3	4	5
halt	0	0						
nop	1	0						
rrmovl **rA**, **rB**	2	0	**rA**	**rB**				
irmovl **V**, **rB**	3	0	F	**rB**	**V**			
rmmovl **rA**, **D**(**rB**)	4	0	**rA**	**rB**	**D**			
mrmovl **D**(**rB**), **rA**	5	0	**rA**	**rB**	**D**			
OPl **rA**, **rB**	6	**fn**	**rA**	**rB**				
jXX **Dest**	7	**fn**	**Dest**					
cmovXX **rA**, **rB**	2	**fn**	**rA**	**rB**				
call **Dest**	8	0	**Dest**					
ret	9	0						
pushl **rA**	A	0	**rA**	F				
popl **rA**	B	0	**rA**	F				

(4-byte values are all little-endian)

Fig 4.2, 4.4

Y86 Datapath Skeleton
May 7, 2015

SWS and Taylor Cathcart, based on a skeleton from Matt Kretchmar



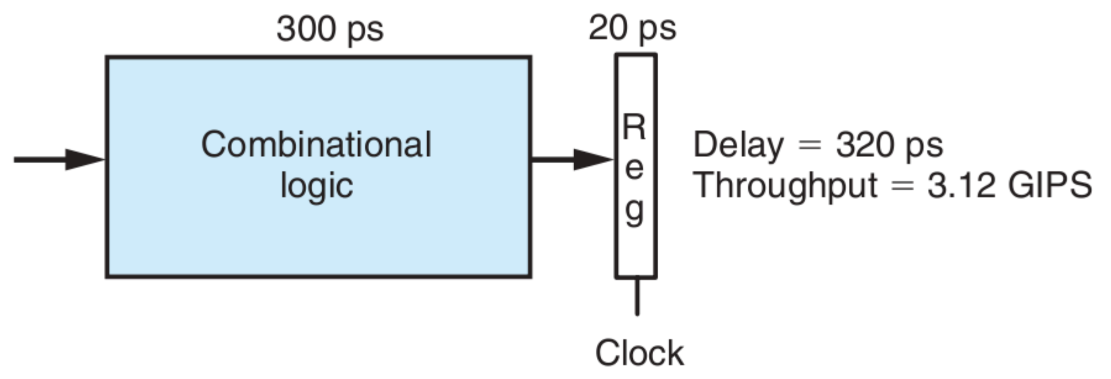
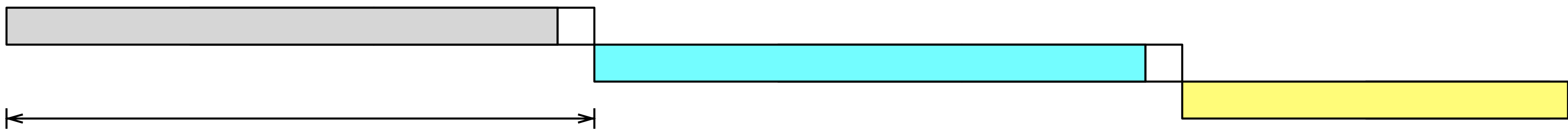
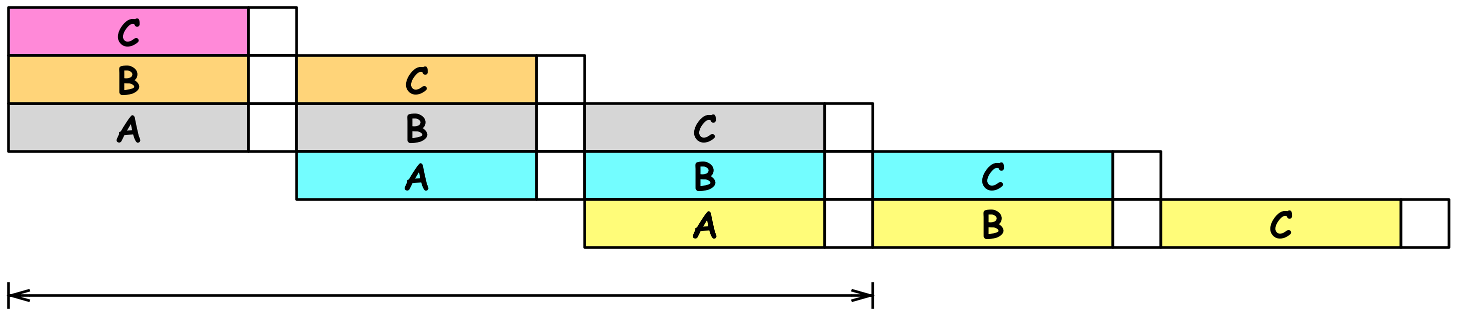


Fig 4.32a





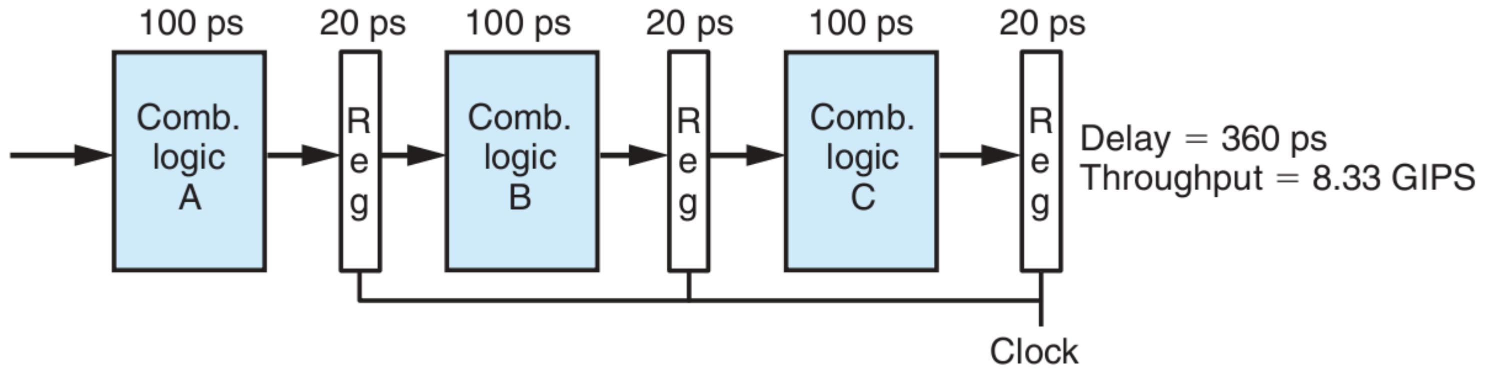
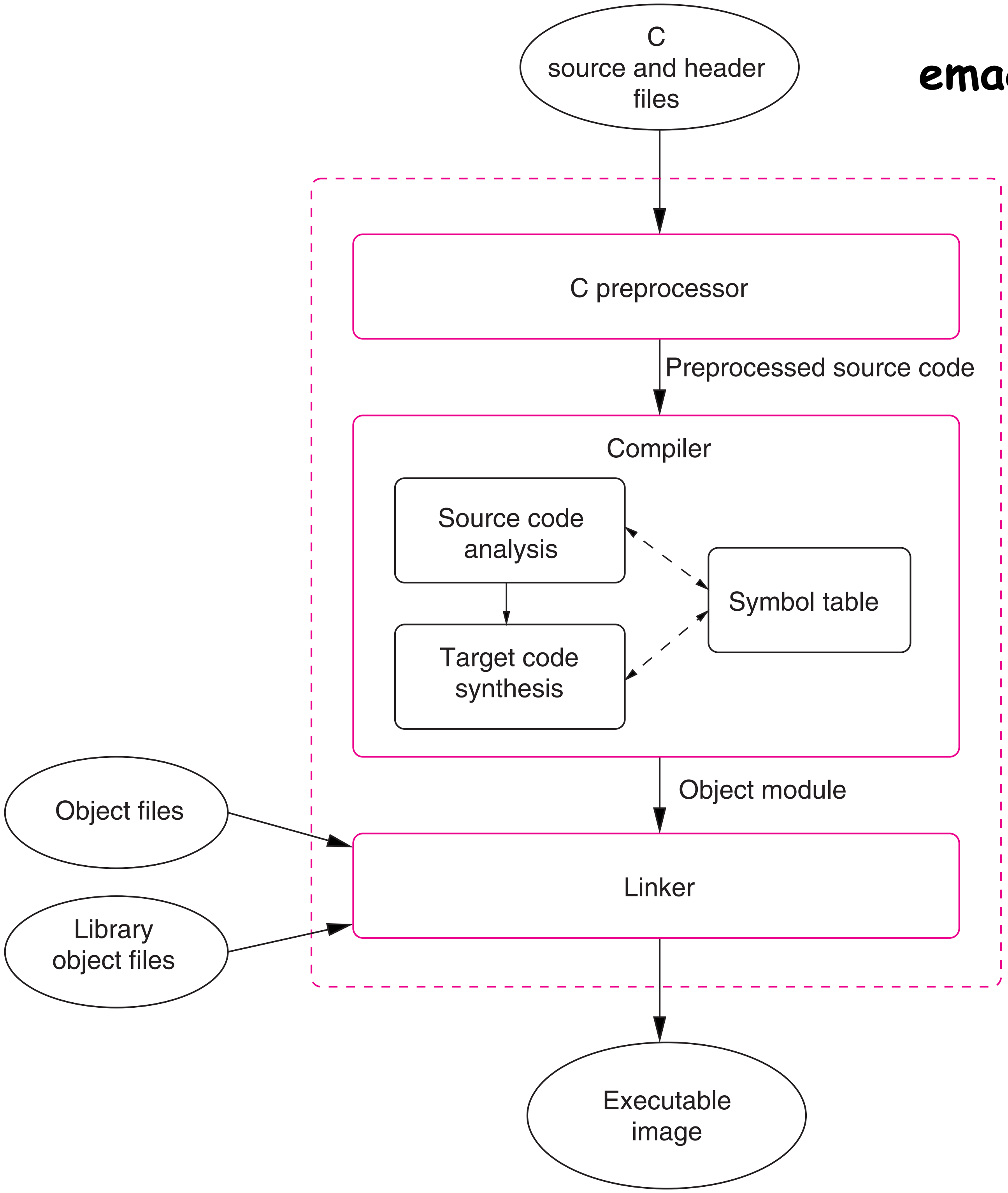


Fig 4.33a

High-Level Languages

wikipedia.org



emacs

gcc

IDE

Patt & Patel, Fig 11-2

www.swissknifeshop.com

C Program Structure

```
/*
 *
 *  Program Name : countdown, our first C program
 *
 *  Description  : This program prompts the user to type in
 *  a positive number and counts down from that number to 0,
 *  displaying each number along the way.
 *
 */

/* The next two lines are preprocessor directives */
#include <stdio.h>
#define STOP 0

/* Function    : main                                     */
/* Description : prompt for input, then display countdown */
int main()
{
  /* Variable declarations */
  int counter;         /* Holds intermediate count values */
  int startPoint;      /* Starting point for count down   */

  /* Prompt the user for input */
  printf("===== Countdown Program =====\n");
  printf("Enter a positive integer: ");
  scanf("%d", &startPoint);

  /* Count down from the input number to 0 */
  for (counter = startPoint; counter >= STOP; counter--)
    printf("%d\n", counter);
}
```

Patt and Patel, Fig 11-3

Basic Data Types

C declaration	32-bit	64-bit
`char`	1	1
`short int`	2	2
`int`	4	4
`long int`	4	8
`long long int`	8	8
`char *`	4	8
`float`	4	4
`double`	8	8

Figure 2.3 **Sizes (in bytes) of C numeric data types.** The number of bytes allocated varies with machine and compiler. This chart shows the values typical of 32-bit and 64-bit machines.

C data type	Minimum	Maximum
`char`	−128	127
`unsigned char`	0	255
`short [int]`	−32,768	32,767
`unsigned short [int]`	0	65,535
`int`	−2,147,483,648	2,147,483,647
`unsigned [int]`	0	4,294,967,295
`long [int]`	−2,147,483,648	2,147,483,647
`unsigned long [int]`	0	4,294,967,295
`long long [int]`	−9,223,372,036,854,775,808	9,223,372,036,854,775,807
`unsigned long long [int]`	0	18,446,744,073,709,551,615

Figure 2.8 **Typical ranges for C integral data types on a 32-bit machine.** Text in square brackets is optional.

```
// sws, cs51, spring 2015

int global = 123;

char c = 'a';

char *string="Green eggs and ham!";

int *gp = &global;

char *cp = &c;

int main() {

  return 0;

}
```

```c
// sws, cs51, spring 2015

int global = 123;

char c = 'a';

char *string="Green eggs and ham!";

int *gp = &global;

char *cp = &c;

int main() {

  return 0;

}
```

```
gcc -g -m32 -O0 -o types types.c
gdb types
```

$\longrightarrow$

```
// sws, cs51, spring 2014

int global = 123;

char c = 'a';

char *string="Green eggs and ham!";

int *gp = &global;

char *cp = &c;

int main() {

  return 0;

}
```

```
gcc -g -m32 -O0 -o types types.c
gdb types
```

$\longrightarrow$

```
(gdb) disas main
Dump of assembler code for function main:
0x00001f80 <main+0>:    push   %ebp
0x00001f81 <main+1>:    mov    %esp,%ebp
0x00001f83 <main+3>:    push   %eax
0x00001f84 <main+4>:    mov    $0x0,%eax
0x00001f89 <main+9>:    movl   $0x0,-0x4(%ebp)
0x00001f90 <main+16>:   add    $0x4,%esp
0x00001f93 <main+19>:   pop    %ebp
0x00001f94 <main+20>:   ret
End of assembler dump.
(gdb) x/32b 0x1f95
0x1f95: 0x47   0x72   0x65   0x65   0x6e   0x20   0x65   0x67
0x1f9d: 0x67   0x73   0x20   0x61   0x6e   0x64   0x20   0x68
0x1fa5: 0x61   0x6d   0x21   0x00   0x01   0x00   0x00   0x00
0x1fad: 0x1c   0x00   0x00   0x00   0x00   0x00   0x00   0x00
(gdb) x/4x 0x2000
0x2000 <global>:     0x7b   0x00   0x00   0x00
(gdb) x/4x 0x2004
0x2004 <c>:    0x61   0x00   0x00   0x00
(gdb) x/4x 0x200C
0x200c <gp>:    0x00   0x20   0x00   0x00
(gdb) x/4x 0x2010
0x2010 <cp>:    0x04   0x20   0x00   0x00
(gdb)
```

```c
// sws, cs51, spring 2015

int global = 123;

char c = 'a';

char *string="Green eggs and ham!";

int *gp = &global;

char *cp = &c;

int main() {

  return 0;

}
```

```
gcc -g -m32 -O0 -o types types.c
gdb types
```

```
(gdb) disas main
Dump of assembler code for function main:
0x00001f80 <main+0>:    push   %ebp
0x00001f81 <main+1>:    mov    %esp,%ebp
0x00001f83 <main+3>:    push   %eax
0x00001f84 <main+4>:    mov    $0x0,%eax
0x00001f89 <main+9>:    movl   $0x0,-0x4(%ebp)
0x00001f90 <main+16>:   add    $0x4,%esp
0x00001f93 <main+19>:   pop    %ebp
0x00001f94 <main+20>:   ret
End of assembler dump.
(gdb) x/32b 0x1f95
0x1f95:     0x47 0x72 0x65 0x65 0x6e 0x20 0x65 0x67
0x1f9d:     0x67 0x73 0x20 0x61 0x6e 0x64 0x20 0x68
0x1fa5:     0x61 0x6d 0x21 0x00 0x01 0x00 0x00 0x00
0x1fad:     0x1c 0x00 0x00 0x00 0x00 0x00 0x00 0x00
(gdb) x/4b 0x2000
0x2000 <global>:     0x7b 0x00 0x00 0x00
(gdb) x/4b 0x2004
0x2004 <c>:     0x61 0x00 0x00 0x00
(gdb) x/4b 0x2008
0x2008 <string>:     0x95 0x1f  0x00 0x00
(gdb) x/4b 0x200C
0x200c <gp>:   0x00 0x20 0x00 0x00
(gdb) x/4b 0x2010
0x2010 <cp>:   0x04 0x20 0x00 0x00
```

```
// sws, cs51, spring 2015

int global = 123;

char c = 'a';

char *string="Green eggs and ham!";

int *gp = &global;

char *cp = &c;

int main() {

  return 0;

}
```

```
gcc -g -m32 -O0 -o types types.c
gdb types
```

$\longrightarrow$

```
(gdb) disas main
Dump of assembler code for function main:
0x00001f80 <main+0>:    push   %ebp
0x00001f81 <main+1>:    mov    %esp,%ebp
0x00001f83 <main+3>:    push   %eax
0x00001f84 <main+4>:    mov    $0x0,%eax
0x00001f89 <main+9>:    movl   $0x0,-0x4(%ebp)
0x00001f90 <main+16>:   add    $0x4,%esp
0x00001f93 <main+19>:   pop    %ebp
0x00001f94 <main+20>:   ret
End of assembler dump.
(gdb) x/32b 0x1f95
0x1f95:   0x47 0x72 0x65 0x65 0x6e 0x20 0x65 0x67
0x1f9d:   0x67 0x73 0x20 0x61 0x6e 0x64 0x20 0x68
0x1fa5:   0x61 0x6d 0x21 0x00 0x01 0x00 0x00 0x00
0x1fad:   0x1c 0x00 0x00 0x00 0x00 0x00 0x00 0x00
(gdb) x/4b 0x2000
0x2000 <global>:   0x7b 0x00 0x00 0x00
(gdb) x/4b 0x2004
0x2004 <c>:   0x61 0x00 0x00 0x00
(gdb) x/4b 0x2008
0x2008 <string>:   0x95 0x1f 0x00 0x00
(gdb) x/4b 0x200C
0x200c <gp>:   0x00 0x20 0x00 0x00
(gdb) x/4b 0x2010
0x2010 <cp>:   0x04 0x20 0x00 0x00
```

```
// sws, cs51, spring 2015

int global = 123;

char c = 'a';

char *string="Green eggs and ham!"

int *gp = &global;

char *cp = &c;

int main() {

  return 0;

}
```

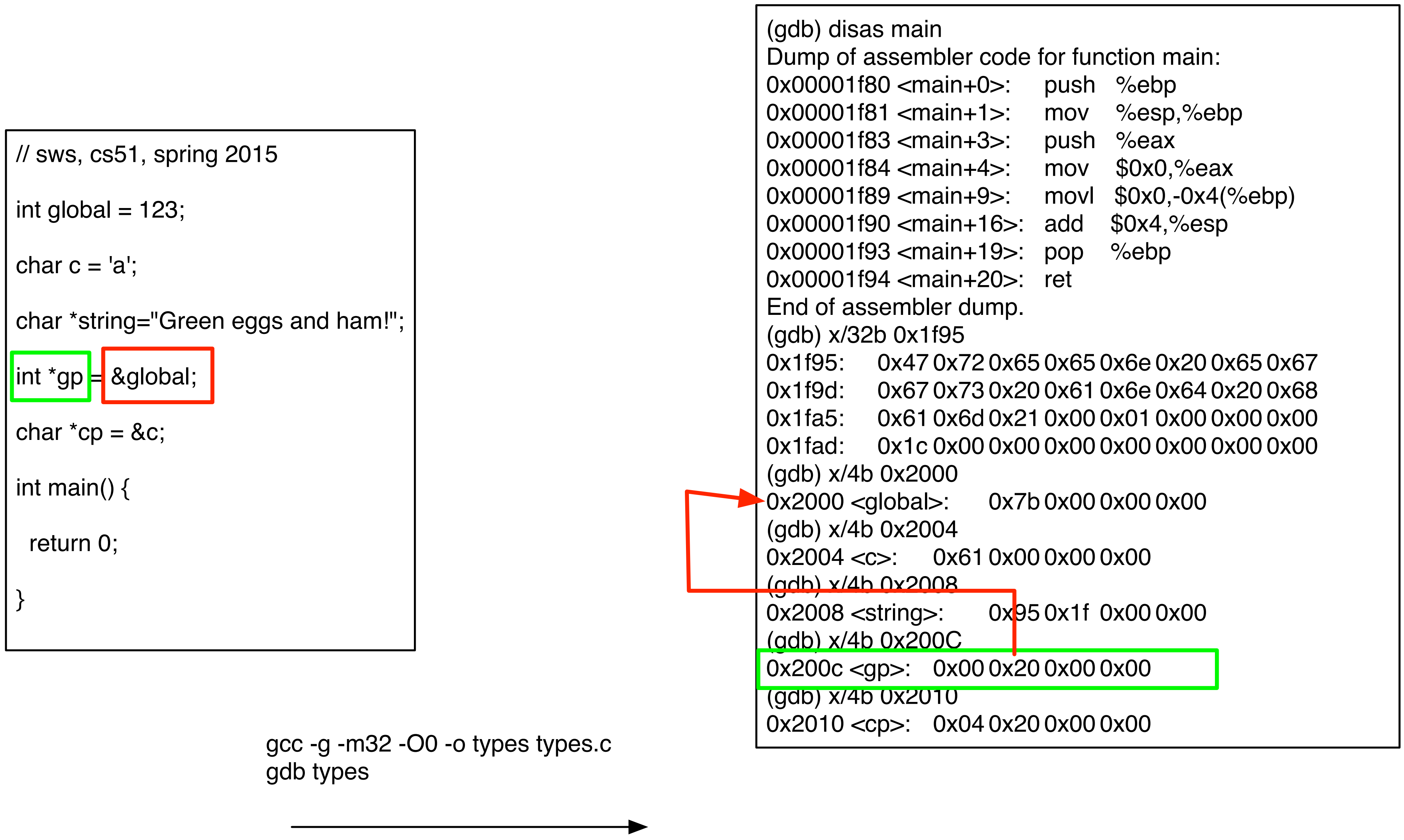
```
gcc -g -m32 -O0 -o types types.c
gdb types
```

```
(gdb) disas main
Dump of assembler code for function main:
0x00001f80 <main+0>:    push   %ebp
0x00001f81 <main+1>:    mov    %esp,%ebp
0x00001f83 <main+3>:    push   %eax
0x00001f84 <main+4>:    mov    $0x0,%eax
0x00001f89 <main+9>:    movl   $0x0,-0x4(%ebp)
0x00001f90 <main+16>:   add    $0x4,%esp
0x00001f93 <main+19>:   pop    %ebp
0x00001f94 <main+20>:   ret
End of assembler dump.
(gdb) x/32b 0x1f95
0x1f95:   0x47 0x72 0x65 0x65 0x6e 0x20 0x65 0x67
0x1f9d:   0x67 0x73 0x20 0x61 0x6e 0x64 0x20 0x68
0x1fa5:   0x61 0x6d 0x21 0x00 0x01 0x00 0x00 0x00
0x1fad:   0x1c 0x00 0x00 0x00 0x00 0x00 0x00 0x00
(gdb) x/4b 0x2000
0x2000 <global>:    0x7b 0x00 0x00 0x00
(gdb) x/4b 0x2004
0x2004 <c>:    0x61 0x00 0x00 0x00
(gdb) x/4b 0x2008
0x2008 <string>:    0x95 0x1f 0x00 0x00
(gdb) x/4b 0x200C
0x200c <gp>:   0x00 0x20 0x00 0x00
(gdb) x/4b 0x2010
0x2010 <cp>:   0x04 0x20 0x00 0x00
```

```c
// sws, cs51, spring 2015

int global = 123;

char c = 'a';

char *string="Green eggs and ham!"

int *gp = &global;

char *cp = &c;

int main() {

  return 0;

}
```

```
gcc -g -m32 -O0 -o types types.c
gdb types
```

```
(gdb) disas main
Dump of assembler code for function main:
0x00001f80 <main+0>:    push   %ebp
0x00001f81 <main+1>:    mov    %esp,%ebp
0x00001f83 <main+3>:    push   %eax
0x00001f84 <main+4>:    mov    $0x0,%eax
0x00001f89 <main+9>:    movl   $0x0,-0x4(%ebp)
0x00001f90 <main+16>:   add    $0x4,%esp
0x00001f93 <main+19>:   pop    %ebp
0x00001f94 <main+20>:   ret
End of assembler dump.
(gdb) x/32b 0x1f95
0x1f95:   0x47 0x72 0x65 0x65 0x6e 0x20 0x65 0x67
0x1f9d:   0x67 0x73 0x20 0x61 0x6e 0x64 0x20 0x68
0x1fa5:   0x61 0x6d 0x21 0x00 0x01 0x00 0x00 0x00
0x1fad:   0x1c 0x00 0x00 0x00 0x00 0x00 0x00 0x00
(gdb) x/4b 0x2000
0x2000 <global>:      0x7b 0x00 0x00 0x00
(gdb) x/4b 0x2004
0x2004 <c>:     0x61 0x00 0x00 0x00
(gdb) x/4b 0x2008
0x2008 <string>:      0x95 0x1f  0x00 0x00
(gdb) x/4b 0x200C
0x200c <gp>:    0x00 0x20 0x00 0x00
(gdb) x/4b 0x2010
0x2010 <cp>:    0x04 0x20 0x00 0x00
```



```c
// sws, cs51, spring 2015

int global = 123;

char c = 'a';

char *string="Green eggs and ham!";

int *gp = &global;

char *cp = &c;

int main() {

  return 0;

}
```

```
gcc -g -m32 -O0 -o types types.c
gdb types
```

```
(gdb) disas main
Dump of assembler code for function main:
0x00001f80 <main+0>:    push   %ebp
0x00001f81 <main+1>:    mov    %esp,%ebp
0x00001f83 <main+3>:    push   %eax
0x00001f84 <main+4>:    mov    $0x0,%eax
0x00001f89 <main+9>:    movl   $0x0,-0x4(%ebp)
0x00001f90 <main+16>:   add    $0x4,%esp
0x00001f93 <main+19>:   pop    %ebp
0x00001f94 <main+20>:   ret
End of assembler dump.
(gdb) x/32b 0x1f95
0x1f95:     0x47 0x72 0x65 0x65 0x6e 0x20 0x65 0x67
0x1f9d:     0x67 0x73 0x20 0x61 0x6e 0x64 0x20 0x68
0x1fa5:     0x61 0x6d 0x21 0x00 0x01 0x00 0x00 0x00
0x1fad:     0x1c 0x00 0x00 0x00 0x00 0x00 0x00 0x00
(gdb) x/4b 0x2000
0x2000 <global>:     0x7b 0x00 0x00 0x00
(gdb) x/4b 0x2004
0x2004 <c>:     0x61 0x00 0x00 0x00
(gdb) x/4b 0x2008
0x2008 <string>:     0x95 0x1f  0x00 0x00
(gdb) x/4b 0x200C
0x200c <gp>:   0x00 0x20 0x00 0x00
(gdb) x/4b 0x2010
0x2010 <cp>:   0x04 0x20 0x00 0x00
```

```c
// sws, cs51, spring 2015

int global = 123;

char c = 'a';

char *string="Green eggs and ham!";

int *gp = &global;

char *cp = &c;

int main() {

  return 0;

}
```

```
gcc -g -m32 -O0 -o types types.c
gdb types
```

```
(gdb) disas main
Dump of assembler code for function main:
0x00001f80 <main+0>:    push   %ebp
0x00001f81 <main+1>:    mov    %esp,%ebp
0x00001f83 <main+3>:    push   %eax
0x00001f84 <main+4>:    mov    $0x0,%eax
0x00001f89 <main+9>:    movl   $0x0,-0x4(%ebp)
0x00001f90 <main+16>:   add    $0x4,%esp
0x00001f93 <main+19>:   pop    %ebp
0x00001f94 <main+20>:   ret
End of assembler dump.
(gdb) x/32b 0x1f95
0x1f95:    0x47 0x72 0x65 0x65 0x6e 0x20 0x65 0x67
0x1f9d:    0x67 0x73 0x20 0x61 0x6e 0x64 0x20 0x68
0x1fa5:    0x61 0x6d 0x21 0x00 0x01 0x00 0x00 0x00
0x1fad:    0x1c 0x00 0x00 0x00 0x00 0x00 0x00 0x00
(gdb) x/4b 0x2000
0x2000 <global>:    0x7b 0x00 0x00 0x00
(gdb) x/4b 0x2004
0x2004 <c>:    0x61 0x00 0x00 0x00
(gdb) x/4b 0x2008
0x2008 <string>:    0x95 0x1f  0x00 0x00
(gdb) x/4b 0x200C
0x200c <gp>:   0x00 0x20 0x00 0x00
(gdb) x/4b 0x2010
0x2010 <cp>:   0x04 0x20 0x00 0x00
```

C to ISA Demo

(a) C code

```c
int exchange(int *xp, int y)
{
    int x = *xp;

    *xp = y;
    return x;
}
```

(b) Assembly code

```
  xp at %ebp+8, y at %ebp+12
    movl    8(%ebp), %edx      Get xp
    By copying to %eax below, x becomes the return value
    movl    (%edx), %eax       Get x at xp
    movl    12(%ebp), %ecx     Get y
    movl    %ecx, (%edx)       Store y at xp
```

Figure 3.6 **C and assembly code for exchange routine body.** The stack set-up and completion portions have been omitted.

```
// the operation of Fig 3.6 in the book, but turned to globals

int *xp;

int y;
int x;

void exchange0() {
  x = *xp;
  *xp = y;
}
```

```c
// the operation of Fig 3.6 in the book, but turned to globals

int *xp;

int y;
int x;

void exchange0() {
  x = *xp;
  *xp = y;
}
```

gcc -O1 -m32 -S exchange0.c

$\longrightarrow$

```
        .section        __TEXT,__text,regular,pure_instructions
        .globl  _exchange0
        .align  4, 0x90
_exchange0:                             ## @exchange0
## BB#0:
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %esi
        calll   L0$pb
L0$pb:
        popl    %eax
        movl    L_xp$non_lazy_ptr-L0$pb(%eax), %ecx
        movl    (%ecx), %ecx
        movl    (%ecx), %edx
        movl    L_x$non_lazy_ptr-L0$pb(%eax), %esi
        movl    %edx, (%esi)
        movl    L_y$non_lazy_ptr-L0$pb(%eax), %eax
        movl    (%eax), %eax
        movl    %eax, (%ecx)
        popl    %esi
        popl    %ebp
        ret

        .comm   _xp,4,2                 ## @xp
        .comm   _x,4,2                  ## @x
        .comm   _y,4,2                  ## @y

        .section        __IMPORT,__pointers,non_lazy_symbol_pointers
L_x$non_lazy_ptr:
        .indirect_symbol        _x
        .long   0
L_xp$non_lazy_ptr:
        .indirect_symbol        _xp
        .long   0
L_y$non_lazy_ptr:
        .indirect_symbol        _y
        .long   0
```

read a var:

movl <address of var>, reg1
movl (reg1), reg2

```
// the operation of Fig 3.6 in the book, but turned to globals

int *xp;

int y;
int x;

void exchange0() {
  x = *xp;
  *xp = y;
}
```

gcc -O1 -m32 -S exchange0.c

```
        .section        __TEXT,__text,regular,pure_instructions
        .globl  _exchange0
        .align  4, 0x90
_exchange0:                             ## @exchange0
## BB#0:
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %esi
        calll   L0$pb
L0$pb:
        popl    %eax
        movl    L_xp$non_lazy_ptr-L0$pb(%eax), %ecx
        movl    (%ecx), %ecx
        movl    (%ecx), %edx
        movl    L_x$non_lazy_ptr-L0$pb(%eax), %esi
        movl    %edx, (%esi)
        movl    L_y$non_lazy_ptr-L0$pb(%eax), %eax
        movl    (%eax), %eax
        movl    %eax, (%ecx)
        popl    %esi
        popl    %ebp
        ret

        .comm   _xp,4,2                 ## @xp
        .comm   _x,4,2                  ## @x
        .comm   _y,4,2                  ## @y

        .section        __IMPORT,__pointers,non_lazy_symbol_pointers
L_x$non_lazy_ptr:
        .indirect_symbol        _x
        .long   0
L_xp$non_lazy_ptr:
        .indirect_symbol        _xp
        .long   0
L_y$non_lazy_ptr:
        .indirect_symbol        _y
        .long   0
```

read a var:

movl <address of var>, reg1
movl (reg1), reg2

```
// the operation of Fig 3.6 in the book, but turned to globals

int *xp;

int y;
int x;

void exchange0() {
  x = *xp;
  *xp = y;
}
```

gcc -O1 -m32 -S exchange0.c

```
        .section        __TEXT,__text,regular,pure_instructions
        .globl  _exchange0
        .align  4, 0x90
_exchange0:                            ## @exchange0
## BB#0:
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %esi
        calll   L0$pb
L0$pb:
        popl    %eax
        movl    L_xp$non_lazy_ptr-L0$pb(%eax), %ecx
        movl    (%ecx), %ecx
        movl    (%ecx), %edx
        movl    L_x$non_lazy_ptr-L0$pb(%eax), %esi
        movl    %edx, (%esi)
        movl    L_y$non_lazy_ptr-L0$pb(%eax), %eax
        movl    (%eax), %eax
        movl    %eax, (%ecx)
        popl    %esi
        popl    %ebp
        ret

        .comm   _xp,4,2                 ## @xp
        .comm   _x,4,2                  ## @x
        .comm   _y,4,2                  ## @y

        .section        __IMPORT,__pointers,non_lazy_symbol_pointers
L_x$non_lazy_ptr:
        .indirect_symbol        _x
        .long   0
L_xp$non_lazy_ptr:
        .indirect_symbol        _xp
        .long   0
L_y$non_lazy_ptr:
        .indirect_symbol        _y
        .long   0
```

read a var:

movl <address of var>, reg1
movl (reg1), reg2

write a var:

movl <address of var>, reg2
movl reg1, (reg2)

```
// the operation of Fig 3.6 in the book, but turned to globals

int *xp;

int y;
int x;

void exchange0() {
  x = *xp;
  *xp = y;
}
```

gcc -O1 -m32 -S exchange0.c

$\longrightarrow$

```
        .section        __TEXT,__text,regular,pure_instructions
        .globl  _exchange0
        .align  4, 0x90
_exchange0:                             ## @exchange0
## BB#0:
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %esi
        calll   L0$pb
L0$pb:
        popl    %eax
        movl    L_xp$non_lazy_ptr-L0$pb(%eax), %ecx
        movl    (%ecx), %ecx
        movl    (%ecx), %edx
        movl    L_x$non_lazy_ptr-L0$pb(%eax), %esi
        movl    %edx, (%esi)
        movl    L_y$non_lazy_ptr-L0$pb(%eax), %eax
        movl    (%eax), %eax
        movl    %eax, (%ecx)
        popl    %esi
        popl    %ebp
        ret

        .comm   _xp,4,2                  ## @xp
        .comm   _x,4,2                  ## @x
        .comm   _y,4,2                  ## @y

        .section        __IMPORT,__pointers,non_lazy_symbol_pointers
L_x$non_lazy_ptr:
        .indirect_symbol        _x
        .long   0
L_xp$non_lazy_ptr:
        .indirect_symbol        _xp
        .long   0
L_y$non_lazy_ptr:
        .indirect_symbol        _y
        .long   0
```

read a var:

movl <address of var>, reg1
movl (reg1), reg2

write a var:

movl <address of var>, reg2
movl reg1, (reg2)

```
// the operation of Fig 3.6 in the book, but turned to globals

int *xp;

int y;
int x;

void exchange0() {
  x = *xp;
  *xp = y;
}
```

gcc -O1 -m32 -S exchange0.c

```
        .section        __TEXT,__text,regular,pure_instructions
        .globl  _exchange0
        .align  4, 0x90
_exchange0:                             ## @exchange0
## BB#0:
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %esi
        calll   L0$pb
L0$pb:
        popl    %eax
        movl    L_xp$non_lazy_ptr-L0$pb(%eax), %ecx
        movl    (%ecx), %ecx
        movl    (%ecx), %edx
        movl    L_x$non_lazy_ptr-L0$pb(%eax), %esi
        movl    %edx, (%esi)
        movl    L_y$non_lazy_ptr-L0$pb(%eax), %eax
        movl    (%eax), %eax
        movl    %eax, (%ecx)
        popl    %esi
        popl    %ebp
        ret

        .comm   _xp,4,2                 ## @xp
        .comm   _x,4,2                  ## @x
        .comm   _y,4,2                  ## @y

        .section        __IMPORT,__pointers,non_lazy_symbol_pointers
L_x$non_lazy_ptr:
        .indirect_symbol        _x
        .long   0
L_xp$non_lazy_ptr:
        .indirect_symbol        _xp
        .long   0
L_y$non_lazy_ptr:
        .indirect_symbol        _y
        .long   0
```

read a var:

movl <address of var>, reg1
movl (reg1), reg2

write a var:

movl <address of var>, reg2
movl reg1, (reg2)

```
// the operation of Fig 3.6 in the book, but turned to globals

int *xp;

int y;
int x;

void exchange0() {
  x = *xp;
  *xp = y;
}
```

gcc -O1 -m32 -S exchange0.c

$\longrightarrow$

```
        .section        __TEXT,__text,regular,pure_instructions
        .globl  _exchange0
        .align  4, 0x90
_exchange0:                             ## @exchange0
## BB#0:
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %esi
        calll   L0$pb
L0$pb:
        popl    %eax
        movl    L_xp$non_lazy_ptr-L0$pb(%eax), %ecx
        movl    (%ecx), %ecx
        movl    (%ecx), %edx
        movl    L_x$non_lazy_ptr-L0$pb(%eax), %esi
        movl    %edx, (%esi)
        movl    L_y$non_lazy_ptr-L0$pb(%eax), %eax
        movl    (%eax), %eax
        movl    %eax, (%ecx)
        popl    %esi
        popl    %ebp
        ret

        .comm   _xp,4,2                 ## @xp
        .comm   _x,4,2                  ## @x
        .comm   _y,4,2                  ## @y

        .section        __IMPORT,__pointers,non_lazy_symbol_pointers
L_x$non_lazy_ptr:
        .indirect_symbol        _x
        .long   0
L_xp$non_lazy_ptr:
        .indirect_symbol        _xp
        .long   0
L_y$non_lazy_ptr:
        .indirect_symbol        _y
        .long   0
```

```
// from Fig 3.6 in the book

int exchange(int *xp, int y) {
  int x = *xp;

  *xp = y;
  return x;
}
```

```
gcc -O1 -m32 -S exchange1.c
```

```c
// from Fig 3.6 in the book

int exchange(int *xp, int y) {
  int x = *xp;

  *xp = y;
  return x;
}
```

$$\xrightarrow{\hspace{6em}}$$

```
gcc -O1 -m32 -S exchange1.c
```

```
	.section	__TEXT,__text,regular,pure_instructions
	.globl	_exchange
	.align	4, 0x90
_exchange:				## @exchange
## BB#0:
	pushl	%ebp
	movl	%esp, %ebp
	movl	8(%ebp), %ecx
	movl	(%ecx), %eax
	movl	12(%ebp), %edx
	movl	%edx, (%ecx)
	popl	%ebp
	ret


.subsections_via_symbols
```

read a var:

movl <address of var>, reg1
movl (reg1), reg2

Address is positive offset from %ebp

```
// from Fig 3.6 in the book

int exchange(int *xp, int y) {
  int x = *xp;

  *xp = y;
  return x;
}
```

```
        .section        __TEXT,__text,regular,pure_instructions
        .globl  _exchange
        .align  4, 0x90
_exchange:                                   ## @exchange
## BB#0:
        pushl   %ebp
        movl    %esp, %ebp
        movl    8(%ebp), %ecx
        movl    (%ecx), %eax
        movl    12(%ebp), %edx
        movl    %edx, (%ecx)
        popl    %ebp
        ret


.subsections_via_symbols
```

$\longrightarrow$

```
gcc -O1 -m32 -S exchange1.c
```

read a var:

movl <address of var>, reg1
movl (reg1), reg2

write a var:

movl <address of var>, reg2
movl reg1, (reg2)

Address is positive offset from %ebp

```
// from Fig 3.6 in the book

int exchange(int *xp, int y) {
  int x = *xp;

  *xp = y;
  return x;
}
```

```
        .section        __TEXT,__text,regular,pure_instructions
        .globl  _exchange
        .align  4, 0x90
_exchange:                              ## @exchange
## BB#0:
        pushl   %ebp
        movl    %esp, %ebp
        movl    8(%ebp), %ecx
        movl    (%ecx), %eax
        movl    12(%ebp), %edx
        movl    %edx, (%ecx)
        popl    %ebp
        ret


.subsections_via_symbols
```

$\longrightarrow$

gcc -O1 -m32 -S exchange1.c

read a var:

movl <address of var>, reg1
movl (reg1), reg2

write a var:

movl <address of var>, reg2
movl reg1, (reg2)

Address is positive offset from %ebp

Where is x??

```
// from Fig 3.6 in the book

int exchange(int *xp, int y) {
  int x = *xp;

  *xp = y;
  return x;
}
```

$\xrightarrow{\hspace{3cm}}$

gcc -O1 -m32 -S exchange1.c

```
        .section        __TEXT,__text,regular,pure_instructions
        .globl  _exchange
        .align  4, 0x90
_exchange:                              ## @exchange
## BB#0:
        pushl   %ebp
        movl    %esp, %ebp
        movl    8(%ebp), %ecx
        movl    (%ecx), %eax
        movl    12(%ebp), %edx
        movl    %edx, (%ecx)
        popl    %ebp
        ret


.subsections_via_symbols
```

read a var:

movl <address of var>, reg1
movl (reg1), reg2

write a var:

movl <address of var>, reg2
movl reg1, (reg2)

Address is positive offset from %ebp

Where is x??

```
// from Fig 3.6 in the book

int exchange(int *xp, int y) {
  int x = *xp;

  *xp = y;
  return x;
}
```

gcc -O0 -m32 -S exchange1.c

```
        .section        __TEXT,__text,regular,pure_instructions
        .globl  _exchange
        .align  4, 0x90
_exchange:                              ## @exchange
## BB#0:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $12, %esp
        movl    12(%ebp), %eax
        movl    8(%ebp), %ecx
        movl    %ecx, -4(%ebp)
        movl    %eax, -8(%ebp)
        movl    -4(%ebp), %eax
        movl    (%eax), %eax
        movl    %eax, -12(%ebp)
        movl    -8(%ebp), %eax
        movl    -4(%ebp), %ecx
        movl    %eax, (%ecx)
        movl    -12(%ebp), %eax
        addl    $12, %esp
        popl    %ebp
        ret


.subsections_via_symbols
```

Variables

GLOBALS	Visible everywhere	Init to 0	Accessed via some master pointer

GLOBALS	Visible everywhere	Init to 0	Accessed via some master pointer
LOCALS	Visible within curly braces	Init to ??	Accessed via negative offset from %ebp

GLOBALS	Visible everywhere	Init to 0	Accessed via some master pointer
LOCALS	Visible within curly braces	Init to ??	Accessed via negative offset from %ebp
			Or kept inside a register

GLOBALS	Visible everywhere	Init to 0	Accessed via some master pointer
LOCALS	Visible within curly braces	Init to ??	Accessed via negative offset from %ebp
			Or kept inside a register
			Or optimized away

GLOBALS	Visible everywhere	Init to 0	Accessed via some master pointer
LOCALS	Visible within curly braces	Init to ??	Accessed via negative offset from %ebp
			Or kept inside a register
			Or optimized away
ARGUMENTS	Visible within function		Accessed via positive offset from %ebp