Computer Architecture

Ch5 – The Assembly Language Level

Nguyen Quoc Dinh, FIT – HUI

HCMC, Aug 2015

Concepts

- A pure assembly language is a language in which each statement produces exactly one machine instruction
  - One-to-one correspondence
- Compiler
  - Translate HLL into a “symbolic” language or a numerical machine language
- Assembler
  - Translate a symbolic representation into a numerical machine language

Writing Code in Assembly Language

- Much more tedious than coding in HLL
- There are more instruction than in a HLL required to perform the same function

Writing Code in Assembly Language

- Why would anyone would ever program in ASM?
  - Performance
    - The final code is faster and more dense
    - Embedded application – where operation rates, memories or registers are at a premium
    - The 10-90 relationship
  - Access to the machine
    - Driver, ISR (interrupt service routine)
    - Somethings usually impossible in high-level language

Making Code Run Faster

- Relationship
  - 1% of the program be responsible for 50% of the execution time
  - 10% of the program be responsible for 90% of the execution time
- Wanna speed up the program? Where to start?
  - The 1%
  - Then the 9%

… but the ASM take longer time to write than HLL

Benchmark Tradeoff

	Programmer-years to produce the program	Program execution time in seconds
Assembly language	50	33
High-level language	10	100
Mixed approach before tuning		
Critical 10%	1	90
Other 90%	9	10
Total	10	100
Mixed approach after tuning		
Critical 10%	6	30
Other 90%	9	10
Total	15	40

Comparison of assembly language and high-level language programming, with and without tuning.

It's about Money

- Program manager's software coding estimates
  - For complete code with design, documentation, coding, and code test completed.
  - HLL: 4-8 lines-of-code (LOC) per hour
  - ASM: 8+ LOC per hour but will require 4-8 times the number of lines
  - While there is at least a 2x speedup in writing code, the number of lines will expand by a factor of from 4x to 8x (resulting in 2x to 4x longer development times)

Above mentioned statistics are from Dr. Bradley J. Bazuin, Western Michigan University

Practical Business Approach

- Steps
  - Prototype in a high-level language
  - Use a process monitor or trace of the execution to determine execution time and flow
  - Rewrite critical or slow segments
- Many companies make two versions
  - the fast time to market one (sell at a loss or for no profit)
  - the slower lower manufacturing cost one (where the profit comes from)

See in Other Places

- Another Similar Relationships in Engineering:
  - HLL to ASM relationship for software
  - VHDL/Verilog Code and Synthesis to custom IC Design and Layout for ICs
  - Faster development and delivery, larger product vs.

    Slower development and delivery, smaller more optimized result

Format of an ASM Language Statement

- Fields: (1) Label (2) Opcode (3) Result and Operand Fields (4) Comment

Label	Opcode	Operands	Comments
FORMULA:	MOV	EAX,I	; register EAX = I
	ADD	EAX,J	; register EAX = I + J
	MOV	N,EAX	; N = I + J
I	DD	3	; reserve 4 bytes initialized to 3
J	DD	4	; reserve 4 bytes initialized to 4
N	DD	0	; reserve 4 bytes initialized to 0

(a)

Computation of N = I + J. (a) Pentium 4.

Format of an ASM Language Statement

- Fields: (1) Label (2) Opcode (3) Result and Operand Fields (4) Comment

Label	Opcode	Operands	Comments
FORMULA	MOVE.L	I, D0	; register D0 = I
	ADD.L	J, D0	; register D0 = I + J
	MOVE.L	D0, N	; N = I + J
I	DC.L	3	; reserve 4 bytes initialized to 3
J	DC.L	4	; reserve 4 bytes initialized to 4
N	DC.L	0	; reserve 4 bytes initialized to 0

(b)

Computation of N = I + J. (b) Motorola 680x0.

Pseudoinstructions (1)

Pseudoinstruction	Meaning
SEGMENT	Start a new segment (text, data, etc.) with certain attributes
ENDS	End the current segment
ALIGN	Control the alignment of the next instruction or data
EQU	Define a new symbol equal to a given expression
DB	Allocate storage for one or more (initialized) bytes
DW	Allocate storage for one or more (initialized) 16-bit (word) data items
DD	Allocate storage for one or more (initialized) 32-bit (double) data items
DQ	Allocate storage for one or more (initialized) 64-bit (quad) data items
PROC	Start a procedure
ENDP	End a procedure
MACRO	Start a macro definition

Some of the pseudoinstructions available in the Pentium 4 assembler (MASM).

Pseudoinstructions (2)

Pseudoinstruction	Meaning
ENDM	End a macro definition
PUBLIC	Export a name defined in this module
EXTERN	Import a name from another module
INCLUDE	Fetch and include another file
IF	Start conditional assembly based on a given expression
ELSE	Start conditional assembly if the IF condition above was false
ENDIF	End conditional assembly
COMMENT	Define a new start-of-comment character
PAGE	Generate a page break in the listing
END	Terminate the assembly program

Some of the pseudoinstructions available in the Pentium 4 assembler (MASM).

Example of Macro and Function

```
 1 #include <stdio.h>
 2 #define swa_mac(a, b) int __x = a; a = b; b=__x
 3 void swa_func(int *a, int *b) {
 4     int x = *a;
 5     *a = *b;
 6     *b = x;
 7 }
 8
 9 int main() {
10     int x = 1;
11     int y = 2;
12     printf("x = %d, y = %d \n", x, y);
13     swa_mac(x, y);
14     printf("x = %d, y = %d \n", x, y);
15     swa_func(&x, &y);
16     printf("x = %d, y = %d \n", x, y);
17 }
```

Macro Definition, Call, Expansion

```
MOV     EAX,P
MOV     EBX,Q
MOV     Q,EAX
MOV     P,EBX

MOV     EAX,P
MOV     EBX,Q
MOV     Q,EAX
MOV     P,EBX
```

(a)

```
SWAP    MACRO
        MOV EAX,P
        MOV EBX,Q
        MOV Q,EAX
        MOV P,EBX
        ENDM

        SWAP

        SWAP
```

(b)

Assembly language code for interchanging P and Q twice.
(a) Without a macro. (b) With a macro.

Macro vs. Procedure

- Do you use a macro or make a procedure call?
  - Procedure calls are done during execution
  - Macros are expanded at time of assembly ... known as a macro call
- The output machine code will not distinguish where a macro was use (you can't tell). After reverse assembly, macros will not exist.
- Macros make the machine code longer (require more memory storage space)
- Macros execute faster as no branching may be required (procedure calls ... and return overhead)

Macro vs. Procedure

Item	Macro call	Procedure call
When is the call made?	During assembly	During program execution
Is the body inserted into the object program every place the call is made?	Yes	No
Is a procedure call instruction inserted into the object program and later executed?	No	Yes
Must a return instruction be used after the call is done?	No	Yes
How many copies of the body appear in the object program?	One per macro call	One

Comparison of macro calls with procedure calls.

Macros with Parameters

```
MOV     EAX,P
MOV     EBX,Q
MOV     Q,EAX
MOV     P,EBX

MOV     EAX,R
MOV     EBX,S
MOV     S,EAX
MOV     R,EBX
```

(a)

```
CHANGE  MACRO P1, P2
        MOV EAX,P1
        MOV EBX,P2
        MOV P2,EAX
        MOV P1,EBX
        ENDM

        CHANGE P, Q

        CHANGE R, S
```

(b)

Nearly identical sequences of statements.
(a) Without a macro. (b) With a macro.

Two-Pass Assemblers (1)

Label	Opcode	Operands	Comments	Length	ILC
MARIA:	MOV	EAX, I	EAX = I	5	100
	MOV	EBX, J	EBX = J	6	105
ROBERTA:	MOV	ECX, K	ECX = K	6	111
	IMUL	EAX, EAX	EAX = I * I	2	117
	IMUL	EBX, EBX	EBX = J * J	3	119
	IMUL	ECX, ECX	ECX = K * K	3	122
MARILYN:	ADD	EAX, EBX	EAX = I * I + J * J	2	125
	ADD	EAX, ECX	EAX = I * I + J * J + K * K	2	127
STEPHANY:	JMP	DONE	branch to DONE	5	129

The instruction location counter (ILC) keeps track of the address where the instructions will be loaded in memory. In this example, the statements prior to MARIA occupy 100 bytes.

Two-Pass Assemblers (2)

Symbol	Value	Other information
MARIA	100	
ROBERTA	111	
MARILYN	125	
STEPHANY	129	

A symbol table for the program in previous slide

Two-Pass Assemblers (3)

Opcode	First operand	Second operand	Hexadecimal opcode	Instruction length	Instruction class
AAA	—	—	37	1	6
ADD	EAX	immed32	05	5	4
ADD	reg	reg	01	2	19
AND	EAX	immed32	25	5	4
AND	reg	reg	21	2	19

A few excerpts from the opcode table for a Pentium 4 assembler

Pass One & Pass Two

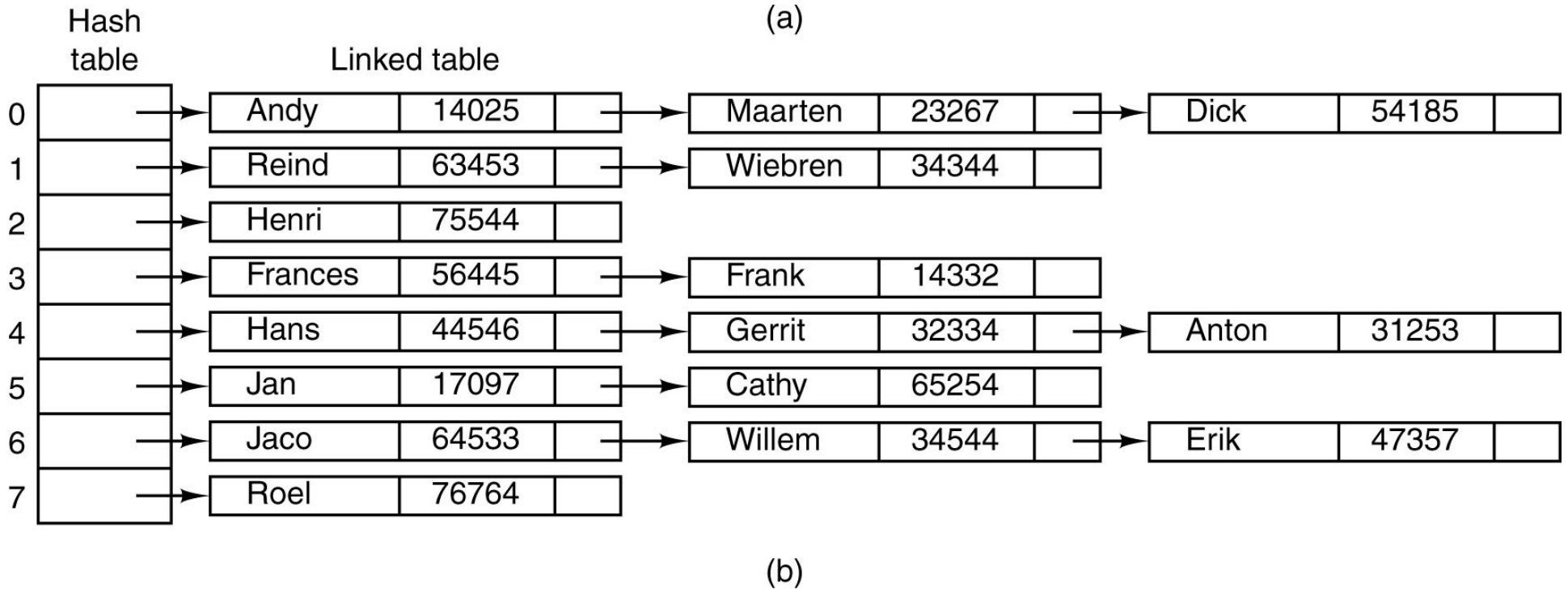
The Symbol Table (1)

Hash coding. (a) Symbols, values, and the hash codes derived from the symbols.

Andy	14025	0
Anton	31253	4
Cathy	65254	5
Dick	54185	0
Erik	47357	6
Frances	56445	3
Frank	14332	3
Gerrit	32334	4
Hans	44546	4
Henri	75544	2
Jan	17097	5
Jaco	64533	6
Maarten	23267	0
Reind	63453	1
Roel	76764	7
Willem	34544	6
Wiebren	34344	1

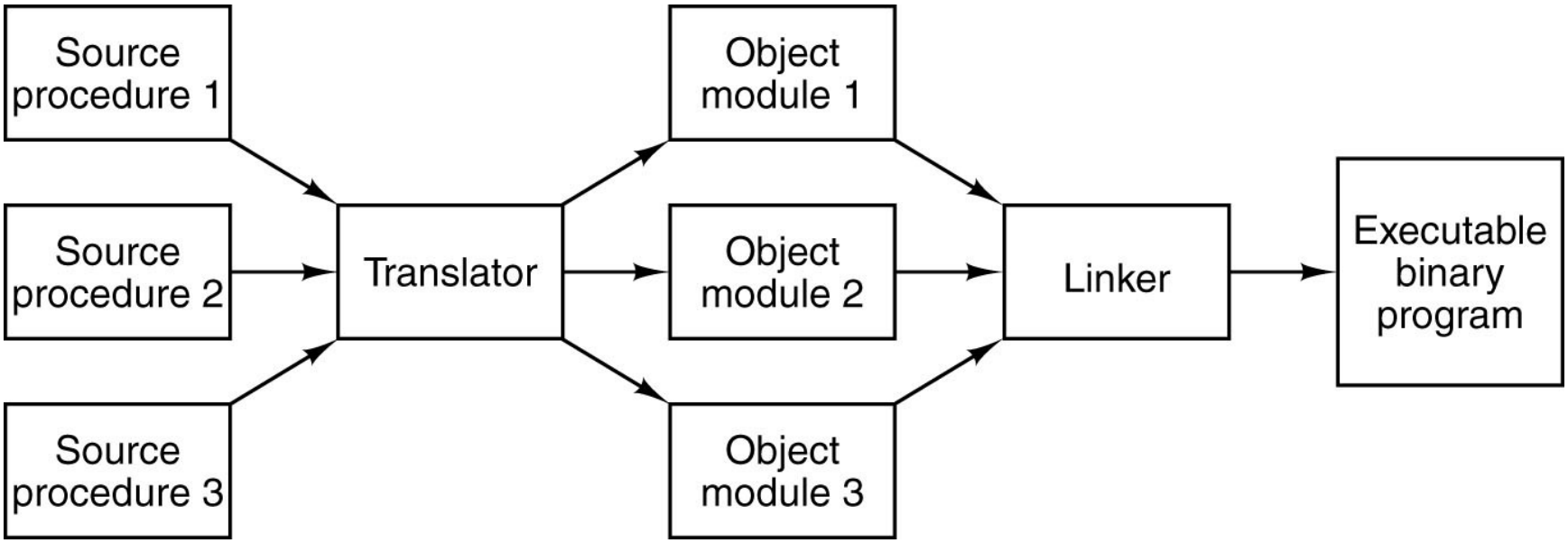
(a)

The Symbol Table (2)



Hash coding. (b)
Eight-entry hash table with linked lists of symbols and values.

Linking and Loading



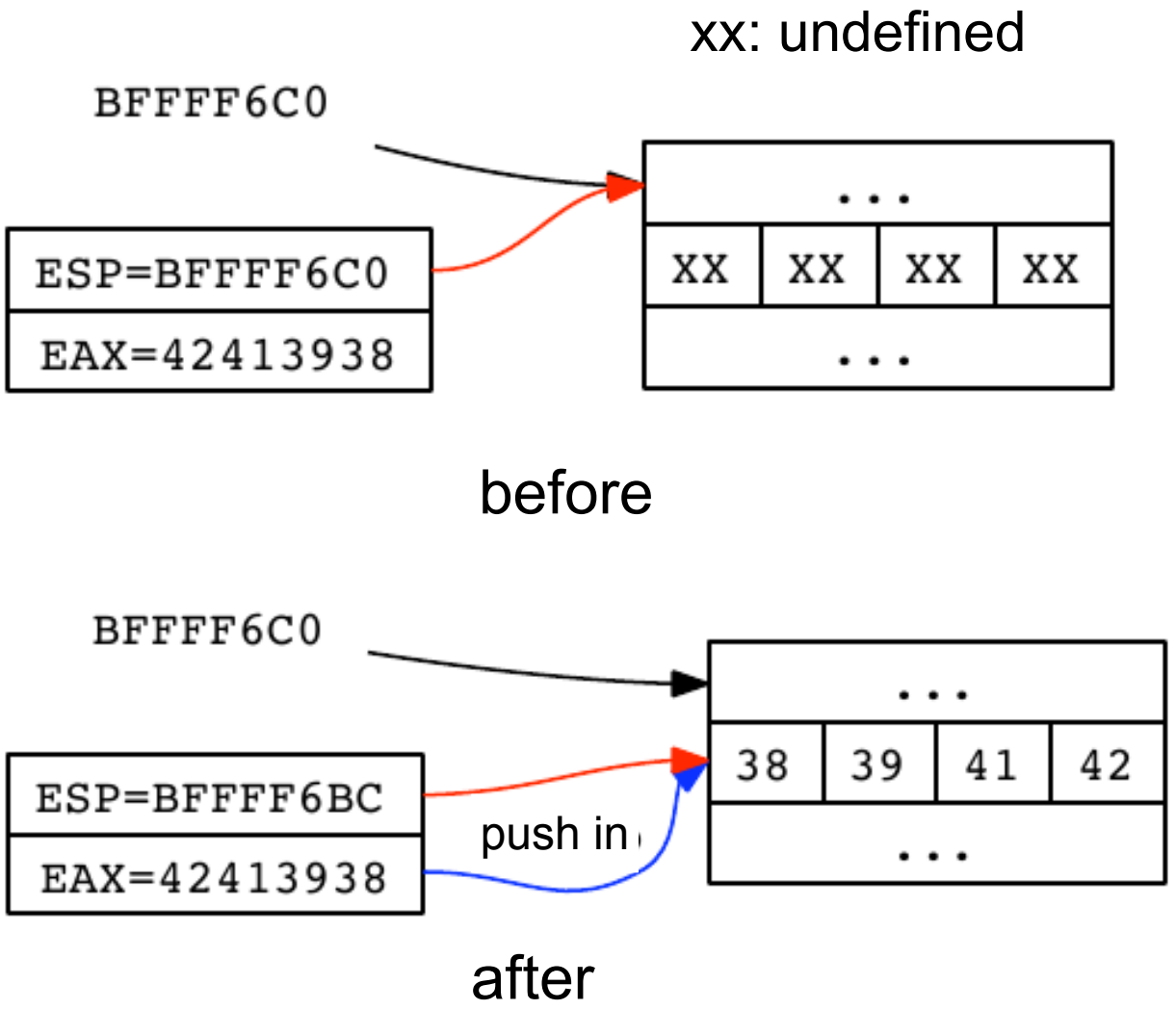
Generation of an executable binary program from a collection of independently translated source procedures requires using a linker.

Linking and Loading

- The source code is translated into object modules.
  - In windows: *.obj
  - In Unix: *.o
- Object modules may be incrementally compiled. They may be loaded from a library. They consist of sufficient “tables” that can be combined to make an executable.
- The resulting executable modules
  - In Windows: *.exe
  - In Unix: *

Stack

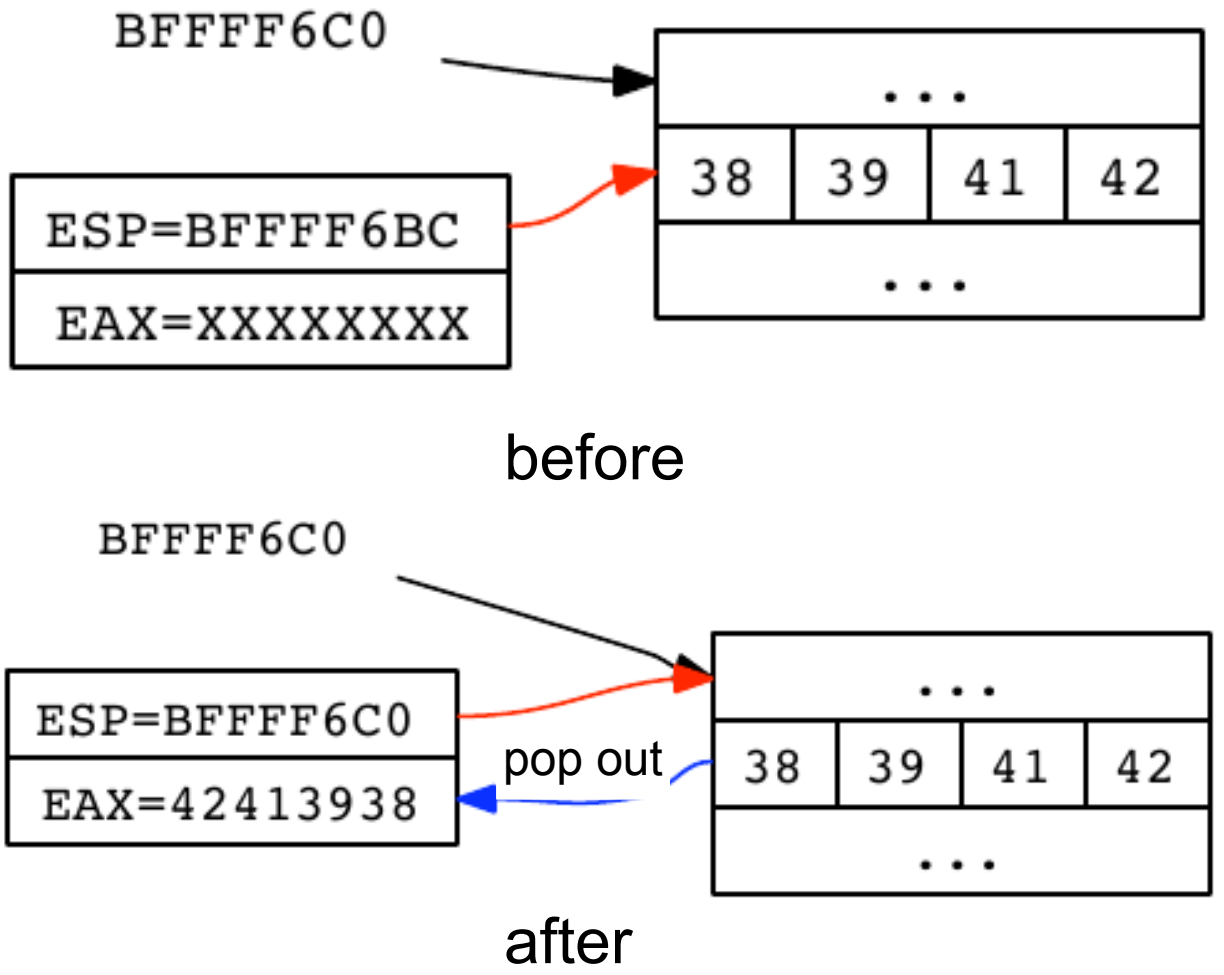
- Push
- Pop



Before and After PUSH EAX command

Stack

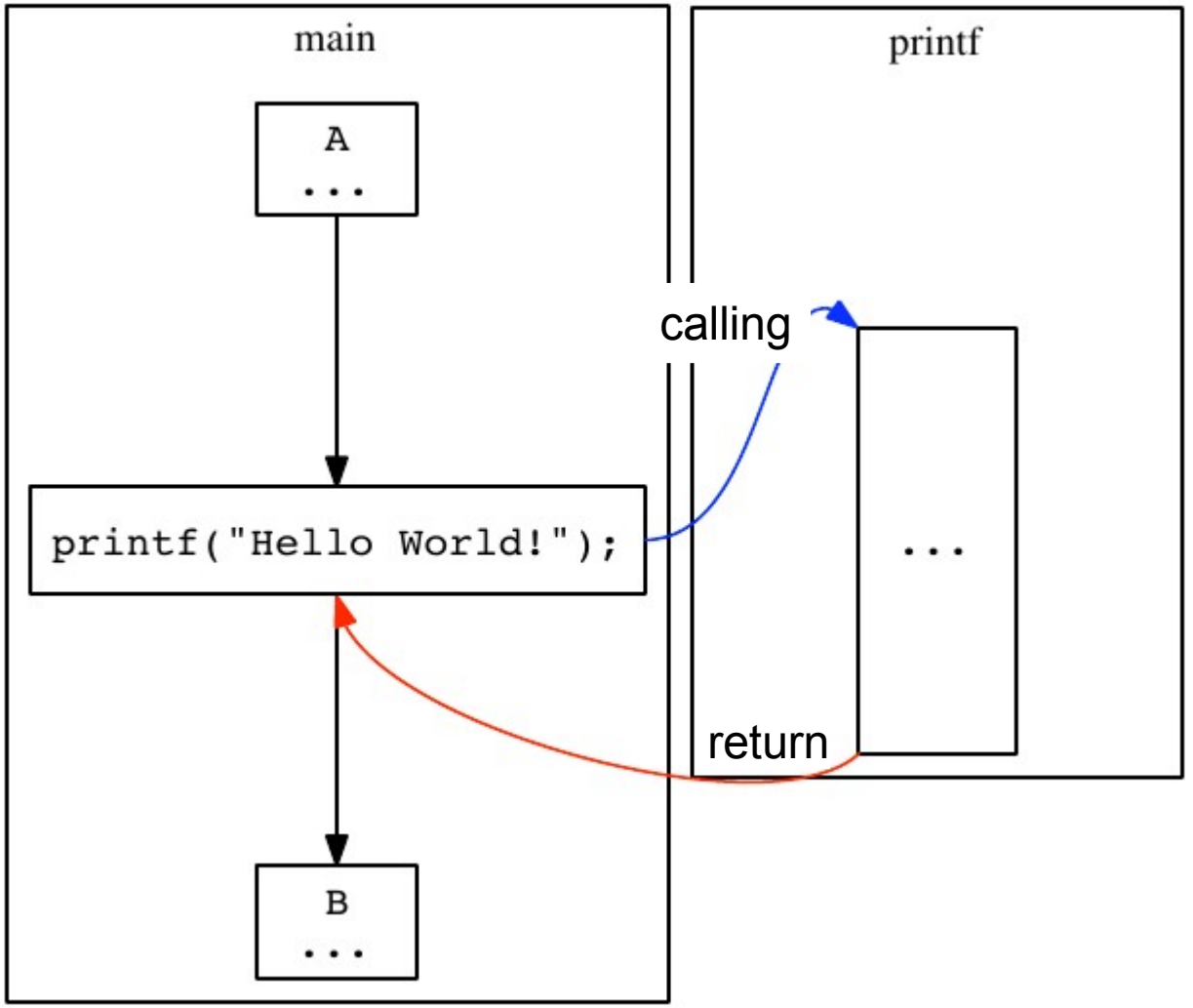
- Push
- Pop



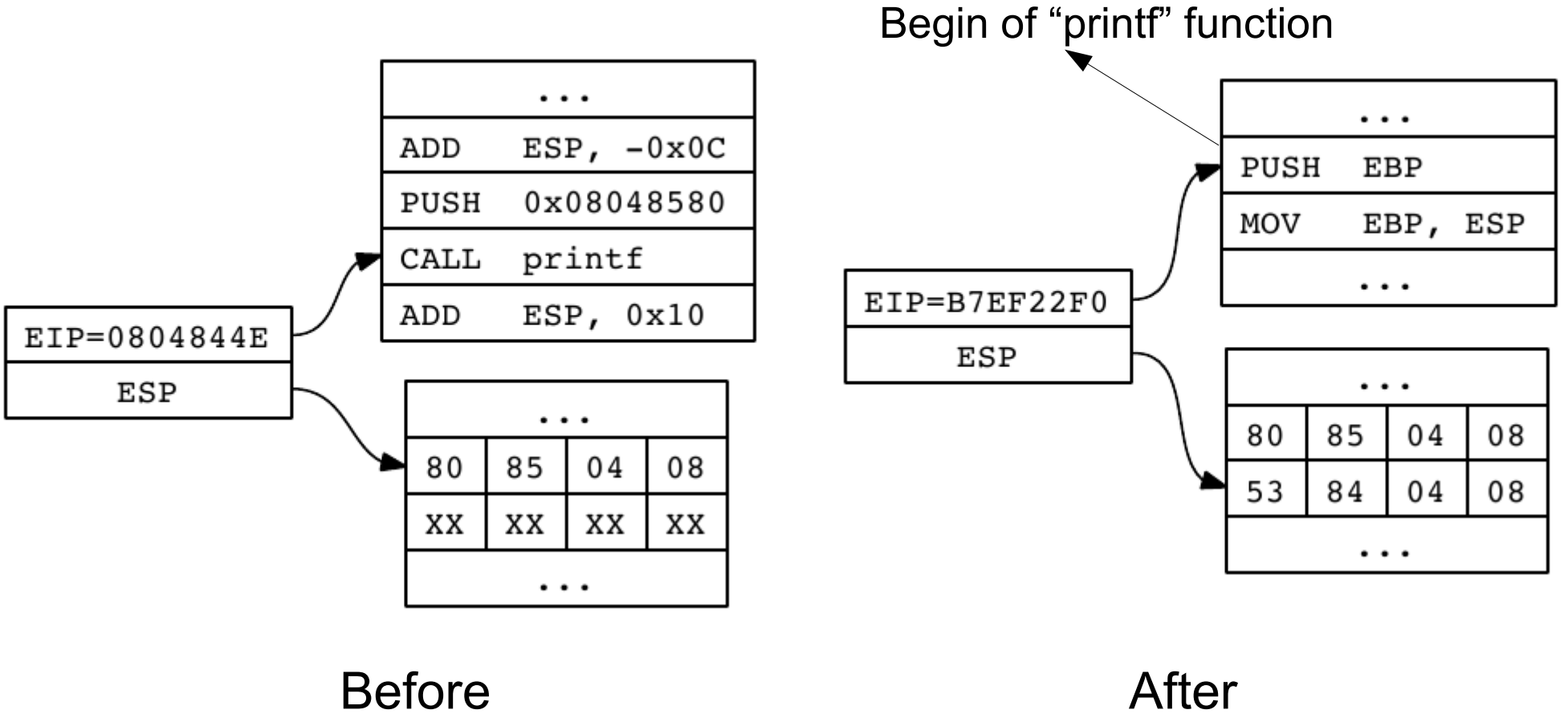
Before and After POP EAX command

Function Calling

```
08048449 PUSH 0x08048580
0804844E CALL printf
08048453 ADD  ESP, 0x10
```

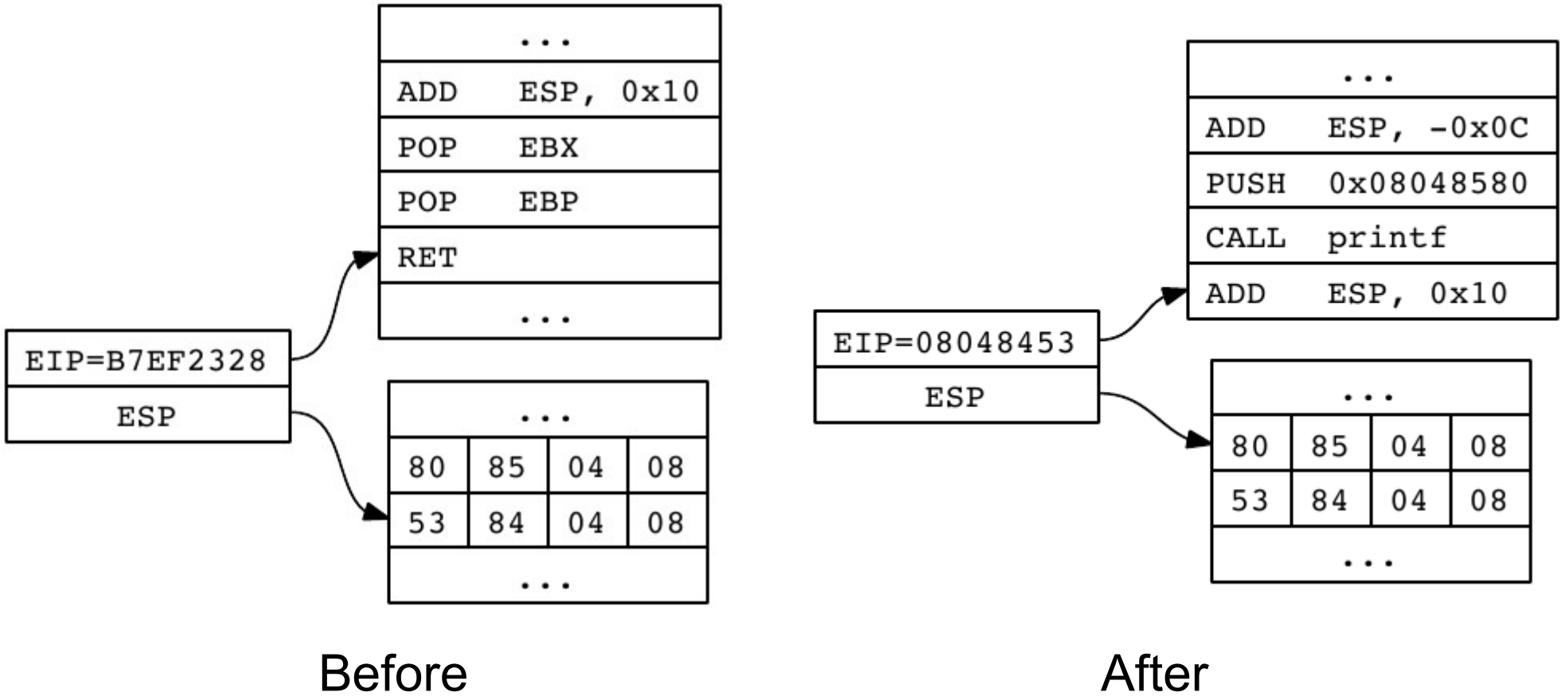


Function Calling



Before and After CALL function

Function Calling



Before and After RET function

Copyright note

- Slides are adopted form lecture notes of Tanenbaum, Structured Computer Organization, Fifth Edition, (c) 2006 Pearson Education, Inc.