COP 3402 Systems Software

Assemblers

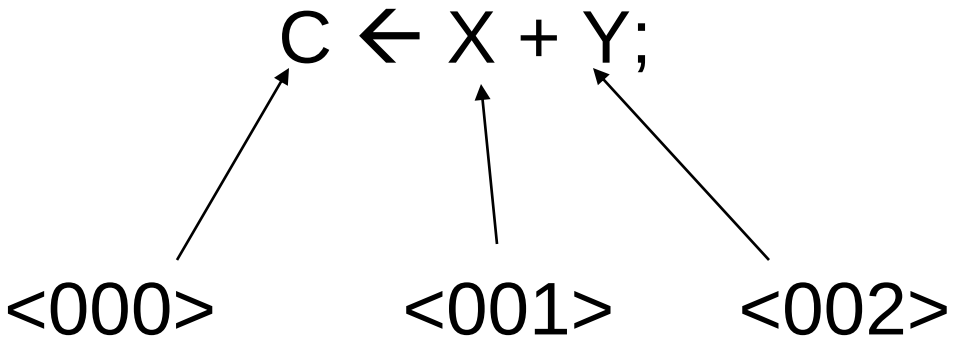
Thanks to Euripides Montagne

ISA
Instruction descriptions

opcode	mnemonic	meaning
0001	LOAD <x>	A ← Mem[x]
0010	ADD <x>	A ← A + Mem[x]
0011	STORE <x>	Mem[x] ← A
0100	SUB <x>	A ← A – Mem[x]
0101	IN <Device_#>	A ← read from Device
0110	OUT <Device_#>	A → output to Device
0111	HALT	Stop
1000	JMP <x>	PC ← x
1001	SKIPZ	If Z = 1 Skip next instruction
1010	SKIPG	If G = 1 Skip next instruction
1011	SKIPN	If L = 1 Skip next instruction

Assembly language Programming examples

Assign a memory location to each variable:



If necessary to use temporary memory locations, assign labels (names) to them.

Assembly language Programming examples

Memory

000	1245
001	1755
002	0000
003	Load <000>
004	Add <001>
005	Store <002>
006	Halt

Memory

000	1245
001	1755
002	3000
003	Load <000>
004	Add <001>
005	Store <002>
006	Halt

One Address Architecture

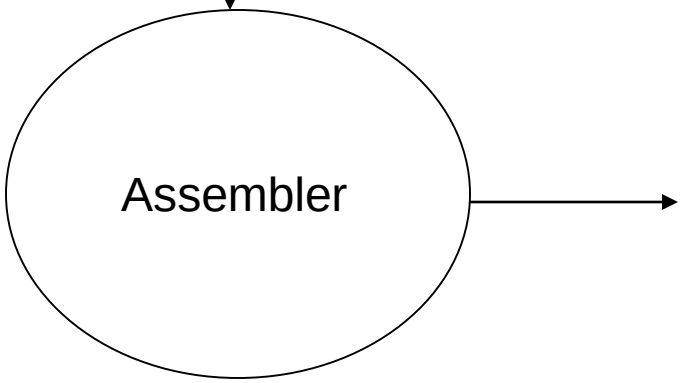
- The instruction format of this one-address architecture consists of 16 bits: 4 bits to represent instructions and 12 bits for addresses :

OP	ADDRESS
0001	**0000 0001 0001**

Assembler: translate Symbolic code to object code(binary)

Assembly Language
003 **Load <000>**
004 Add <001>
005 Store <002>
006 Halt

01 → LOAD
02 → ADD
03 → STORE
04 → SUB
05 → IN
06 → OUT
07 → HALT
08 → JMP
09 → SKIPZ



In binary

003	**0001 0000 0000 0000**
004	0010 0000 0000 0001
005	0011 000000000011
006	0111 000000000000

Assembler Directives

- The next step to improve our assembly language is the incorporation of *pseudo-ops (assembler directives)* to invoke *a special service from the assembler (pseudo-operations do not generate code)*

  .begin → tells the assembler where the program starts

  .data → to reserve a memory location.

  .end → tells the assembler where the program ends.

  Labels are symbolic names used to identify memory locations.

Assembler Directives

This is an example of the usage of assembler directives

.begin

"Assembly language instructions"

halt *(return to OS)*
.data *(to reserve a memory location)*

.end **(** *tells the assembler where the program ends)*

note:
the directive .end can be used to indicate where the program Starts (for eample: ".end <insert label here>"

Assembly language Programming Example 1

Label	opcode	address
start	.begin	
	in	x005
	store	a
	in	x005
	store	b
	load	a
	sub	TWO
	add	b
	out	x009
	halt	
a	.data	0
b	.data	0
TWO	.data	2
	.end	start

Text section (code)

Data section

Assembly language Programming Example 2

	Label	opcode	address
01		; This is	
02		; a comment	
03	start	.begin	x200
04	here	LOAD	sum
05		ADD	a
06		STORE	sum
07		LOAD	b
08		SUB	one
09		STORE	b
0A		SKIPZ	
0B		JMP	here
0C		LOAD	sum
0D		HALT	
0E	sum	.data	x000
0F	a	.data	x005
10	b	.data	x003
11	one	.data	x001
12		.end	start

This program is computing 5 x 3.

ASSEMBLER
Pass 1

	Label	opcode	address	
01		; This is		
02		; a comment		
03	start	.begin	x200	
04	here	LOAD	sum	x200
05		ADD	a	x201
06		STORE	sum	x202
07		LOAD	b	x203
08		SUB	one	x204
09		STORE	b	x205
0A		SKIPZ		x206
0B		JMP	here	x207
0C		LOAD	sum	x208
0D		HALT		x209
0E	sum	.data	x000	x20A
0F	a	.data	x005	x20B
10	b	.data	x003	x20C
11	one	.data	x001	x20D
12		.end	start	x20E

Symbol Table

symbol	address
here	x200
sum	x20A
a	x20B
b	x20C
one	x20D

In pass one the assembler examines the program line by line in order to built the symbol table.

There is an entry in the symbol table for each label found in the program.

Opcode and Symbol Tables

Opcode table

opcode	mnemonic
0001	LOAD
0010	ADD
0011	STOR
0100	SUB
0101	IN
0110	OUT
0111	HALT
1000	JMP
1001	SKIPZ
1010	SKIPG
1011	SKIPN

Symbol Table

symbol	address
here	x200
sum	x20A
a	x20B
b	x20C
one	x20D

Using the symbol table and the opcode table the assembler translates the program to object code.

As the program can be loaded anywhere in memory PC-relative addressing is used to resolve the symbols.

For instance, the offset between LOAD sum and the declaration of sum is 9, because when LOAD sum is fetched for execution, the pc is pointing to the instruction ADD a. (pc + offset = 9)

ASSEMBLER
Pass 2

	Label	opcode	address
01		; This is	
02		; a comment	
03	start	.begin	x200
04	here	LOAD	sum
05		ADD	a
06		STORE	sum
07		LOAD	b
08		SUB	one
09		STORE	b
0A		SKIPZ	
0B		JMP	here
0C		LOAD	sum
0D		HALT	
0E	sum	.data	x000
0F	a	.data	x005
10	b	.data	x003
11	one	.data	x001
12		.end	start

Object code

	x200	0001000000001001 (9 is the offset)
PC →	x201	0010000000001001
	x202	0011000000000111 (7 is the offset)
	x203	0001000000001000
	x204	0100000000001000
	x205	0011000000000110
	x206	
	x207	
	x208	
	x209	
	x20A	
	x20B	
	x20C	
	x20D	
	x20E	

offset

All addresses are pc-relative addresses. (PC + offset)

Recall: PC is always pointing to the next instruction to be fetch.

Assembly language Programming object code

	Label	opcode	address
01		; This is	
02		; a comment	
03	start	.begin	x200
04	here	LOAD	sum
05		ADD	a
06		STORE	sum
07		LOAD	b
08		SUB	one
09		STORE	b
0A		SKIPZ	
0B		JMP	here
0C		LOAD	sum
0D		HALT	
0E	sum	.data	x000
0F	a	.data	x005
10	b	.data	x003
11	one	.data	x001
12		.end	start

Object code

	x200	0001000000001001 (9 is the offset)	
PC →	x201	0010000000001001	
	x202	0011000000000111 (7 is the offset)	
	x203	0001000000001000	
offset	x204	0100000000001000	
	x205	0011000000000110	
	x206	1001000000000000	
	x207	1000111111111000	(-7)
	x208	0001000000000001	
	x209	0111000000000000	
	x20A	0000000000000000	
	x20B	0000000000000101	
	x20C	0000000000000011	
	x20D	0000000000000001	
	x20E		

One's complement

ASSEMBLER
object code

The object code file has several sections:

Header section: Size of code, name source file, size of data

Text section (code): Object code

Data section: Data (in binary)

Relocation information section: Addresses to be fixed up by the linker

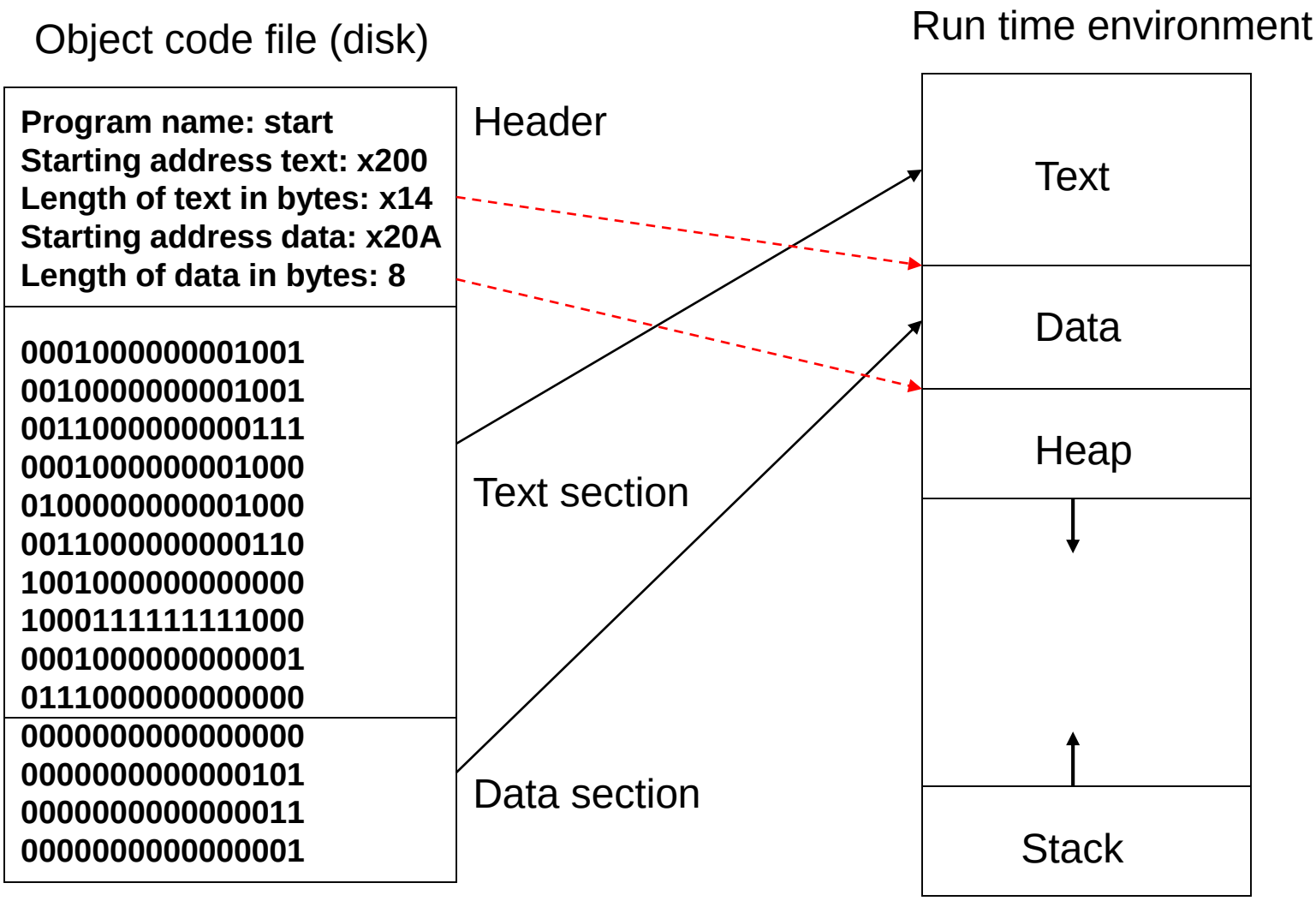
Symbol table section: Global symbols in the program, Imported symbols

Debugging section: Source file and line number information, description of data structures.

ASSEMBLER
object code file for the example

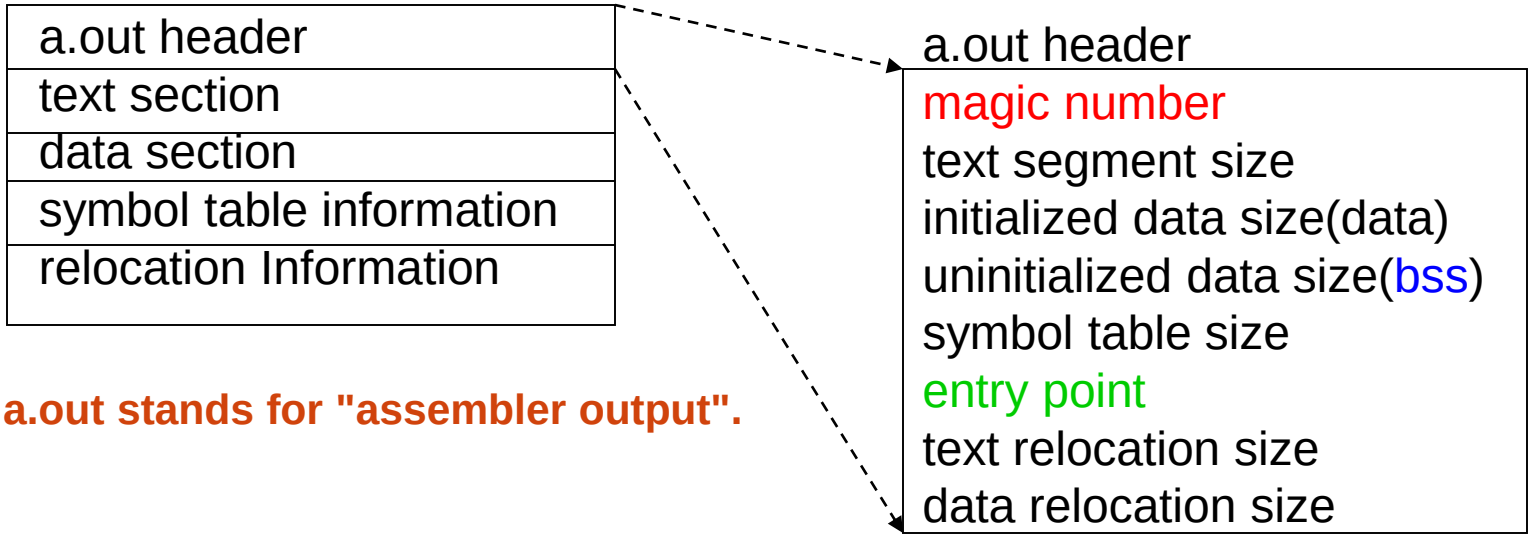
Program name: start **Starting address text: x200** **Length of text in bytes: x14** **Starting address data: x20A** **Length of data in bytes: 8**	Header
0001000000001001 **0010000000001001** **0011000000000111** **0001000000001000** **0100000000001000** **0011000000000110** **1001000000000000** **1000111111111000** **0001000000000001** **0111000000000000**	Text section
0000000000000000 **0000000000000101** **0000000000000011** **0000000000000001**	Data section

Loading Object code in Memory



UNIX a.out format

a.out object code format



a.out stands for "assembler output".

magic number indicates type of executable file.

bss is an acronym for block storage start.

entry point: starting address of the program