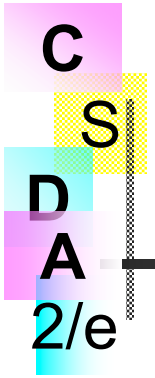
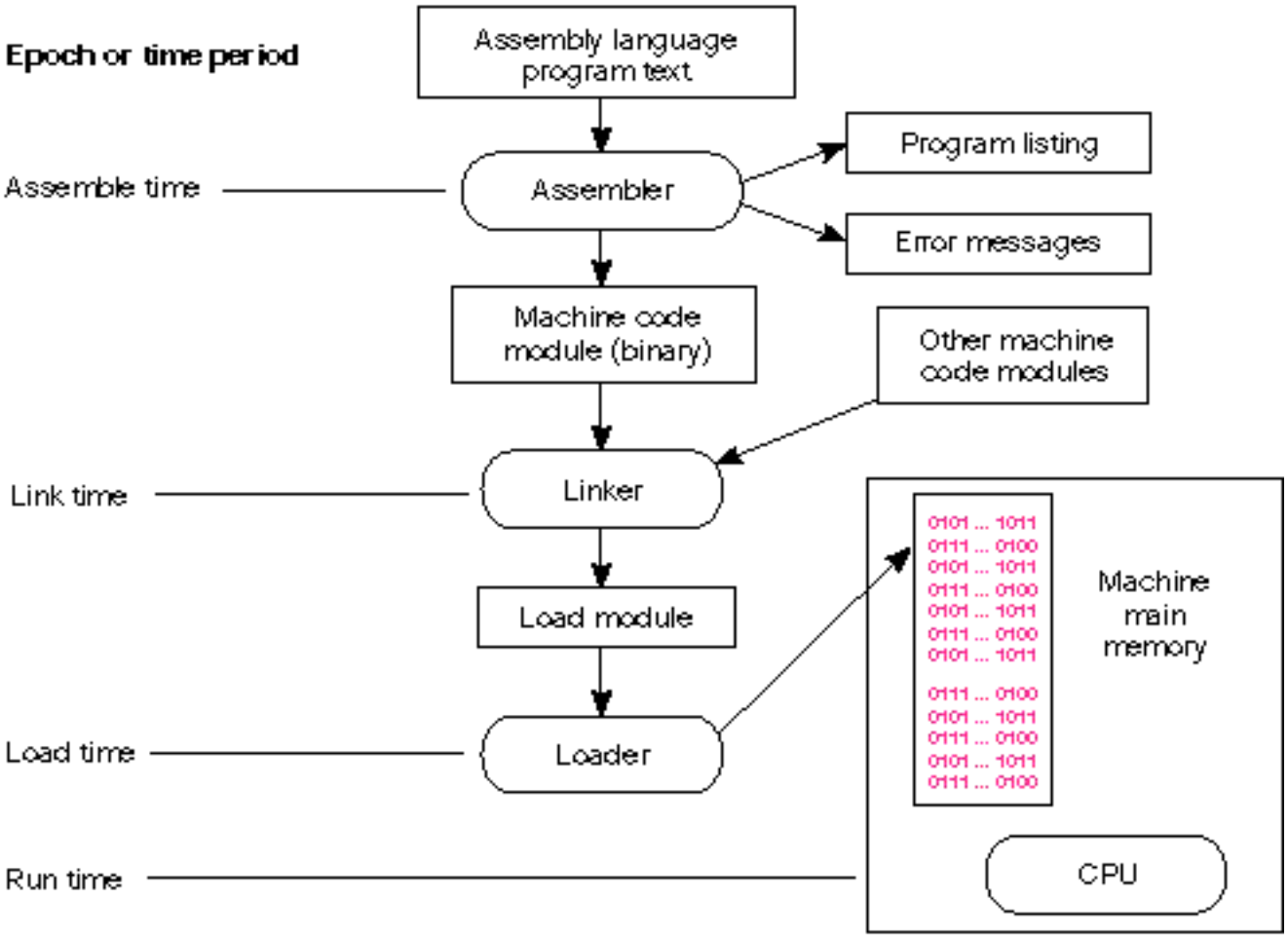


Assembly and Assemblers

- What is an assembler?
  - Translates from assembly language to machine language
- Assembly language structure
- Tasks of the assembler

Fig C.1 Time Periods of Various Processes in Program Development



The Assembler Provides—

- Access to all the machine's resources by the assembled program. This includes access to the entire instruction set of the machine.
- A means for specifying run-time locations of program and data in memory.
- Provide symbolic labels for the representation of constants and addresses.
- Perform assemble-time arithmetic.
- Provide for the use of any synthetic instructions.
- Emit machine code in a form that can be loaded and executed.
- Report syntax errors and provide program listings
- Provide an interface to the module linkers and program loader.
- Expand programmer defined macro routines.

Assembler Syntax and Directives

- Syntax: Label OPCODE Op1, Op2, ... ;Comments
- Pseudo Operations (sometimes called “pseudos,” or directives are “Opcodes” that are actually instructions to the assembler, and that do not result in code being generated.
- Assembler maintains several data structures
  - Table that maps text of opcodes to op number and instruction format(s)
  - “Symbol Table” that maps defined symbols to their value

Table C.1 Assembler Directives

Directive Type	Example	Action

Define a symbolic constant	`NOV .equ 11`
	`MAY .equ "May"`
	`Size .equ 10`
		Set the symbol NOV equal to the constant 11.
		Set the symbol MAY equal to the string "May".
		Set the symbol Size equal to the constant10.

Fix a memory location	`Start .org 0x2000` `main: ld r0, Size`	Fix the location at which the following program or data word will load to 2000_{16}. Begin program execution at this location.

Reserve a block of storage	`Array: .dcb 20`
	`Harray: .dch Size`
	`Warray .dc 20`
		Reserve space for 20 bytes. Base address is Array.
		Reserve space for 10 halfwords. Base address is Harray.
		Reserve space for 20 words. Base address is Warray.

Initialize memory location(s)	`Minus1: .dch 0xffff`
	`Colors: .dc 0, 1, 2, 3`
	`Hi .dcb "Hello"`
		Reserve a halfword at location Minus1; initialize it to the hexadecimal value ffff.
		Reserve space at location Colors for 4 words, and initialize them to 0, 1, 2, and 3.
		Reserve space at location Hi for 5 bytes; init. to 'Hello'

Describe† module linkage	`Out: ... PUBLIC` `... Val EXTERN`	Make the value of Out available to the linker for linkage to other assembled modules. The value of Val is defined externally in another module.

†Not available in SRC, but available in production assemblers.

Figure C.2 Example Program fib.asm

```
; fib.asm. Compute Fibonacci numbers.
; The Fibonacci sequence is defined as follows:
; fib(1) = 1, fib(2) = 1,
; fib(n) = fib(n-1) + fib(n-2) n > 2.
cnt:    .equ 8          ; No. to compute after first two
        .org 0          ; Store sequence at addr. 0
seq:    .dc 1, 1        ; Init. the first two Fib. Nos.
        .dw cnt         ; Storage for the next 8 Fib. Nos.
        .org 0×1000     ; Begin ass'y. at hex. addr. 1000
main:   lar r31, loop   ; Pgm start. Init. branch address
        la r0, cnt      ; Init. count
        la r1, 0        ; Init r1 to index of seq[0]
        la r2, 4        ; Init r2 to index of seq[1]
loop:   ld r3, seq(r1) ; Get fib(n-2)
        addi r1, r1, 4 ; Increment index
        ld r4, seq(r1) ; Get fib(n-1)
        add r3, r3, r4 ; compute fib(n)
        addi r2, r2, 4 ; fib(n) = fib(n-1) + fib(n-2)
        st r3, seq(r2) ; Store fib(n)
        addi r0, r0, -1; Decrement count
        brnz r31, r0   ; loop untill done
        stop
```

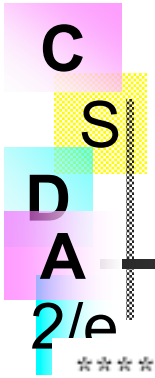


Figure C.3 Assembler listing file, fib.lst

```
*****SRC Assembler***** (SRCTools Version 2.1.0)

 HexLoc     DecLoc     MachWord  Label      Source Code           Comments
00000000  0000000000  00000000          ; fib.asm. Compute Fibonacci numbers.
00000000  0000000000  00000000          ; The Fibonacci sequence is defined as follows:
00000000  0000000000  00000000          ; fib(1) = 1, fib(2) = 1,
00000000  0000000000  00000000          ; fib(n) = fib(n-1) + fib(n-2) n > 2.
00000000  0000000000  00000000  cnt:    .equ 8              ; No. to compute after first two
00000000  0000000000  00000000          .org 0              ; Store sequence at addr. 0
00000000  0000000000  00000001  seq:    .dc 1               ; Init the first two numbers
00000004  0000000004  00000001          .dc 1               ;
00000008  0000000008  00000000          .dw cnt             ; Storage for the next 8 Fib. Nos.
00000028  0000000040  00000000          .org 0×1000         ; Begin ass'y. at hex. addr. 1000
00001000  0000004096  37c0000c  main:   lar r31, loop       ; Pgm start: init branch address
00001004  0000004100  28000008          la r0, cnt          ; Init. count
00001008  0000004104  28400000          la r1, 0            ; Init r1 to index of seq[0]
0000100c  0000004108  28800004          la r2, 4            ; Init r2 to index of seq[1]
00001010  0000004112  08c20000  loop:   ld r3, seq(r1)      ; Get fib(n-2)
00001014  0000004116  68420004          addi r1, r1, 4      ; Increment index
00001018  0000004120  09020000          ld r4, seq(r1)      ; Get fib(n-
1)
0000101c  0000004124  60c64000          add r3, r3, r4      ; compute fib(n)
00001020  0000004128  68840004          addi r2, r2, 4      ; fib(n) = fib(n-1) + fib(n-2)
00001024  0000004132  18c40000          st r3, seq(r2)      ; Store fib(n)
00001028  0000004136  6801ffff          addi r0, r0, -1     ; Decrement count
0000102c  0000004140  403e0003          brnz r31, r0        ; loop untill done
00001030  0000004144  f8000000          stop
```

The 2-Pass Assembly Process

- Pass 1:
  - Init. Location Counter (Assemble-Time “PC”) to 0
  - Pass over program text: enter all symbols into symbol table
    - May not be able to map all symbols on first pass
    - (Definition before use is usually allowed)
  - Determine size of each instruction, map to a location
    - Uses pattern matching to relate opcode to pattern
    - Increment location counter by size
    - Change Location Counter in response to ORG pseudos.
- Pass 2
  - Insert binary code for each opcode and value
  - “Fix up” forward references and variable-sizes instructions
    - Examples include variable-sized branch offsets and constant fields.

Table C.2 Snapshot of the Symbol Table Generated by the Assembler During Pass 1 at Address 0x1008

Key	Symbol	Type	Value	Defined
0	"cnt"	constant	8	defined
1	"seq"	label	00000000	defined
2	"main"	label	00001000	defined
3	"loop"	unknown		undefined