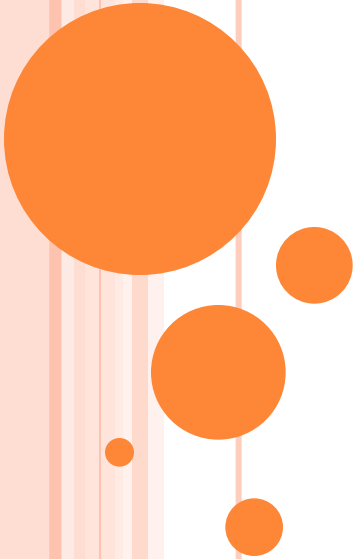
EMBEDDED SOFTWARE DEVELOPMENT



Hardware and software architectures

Hardware and software are intimately related:

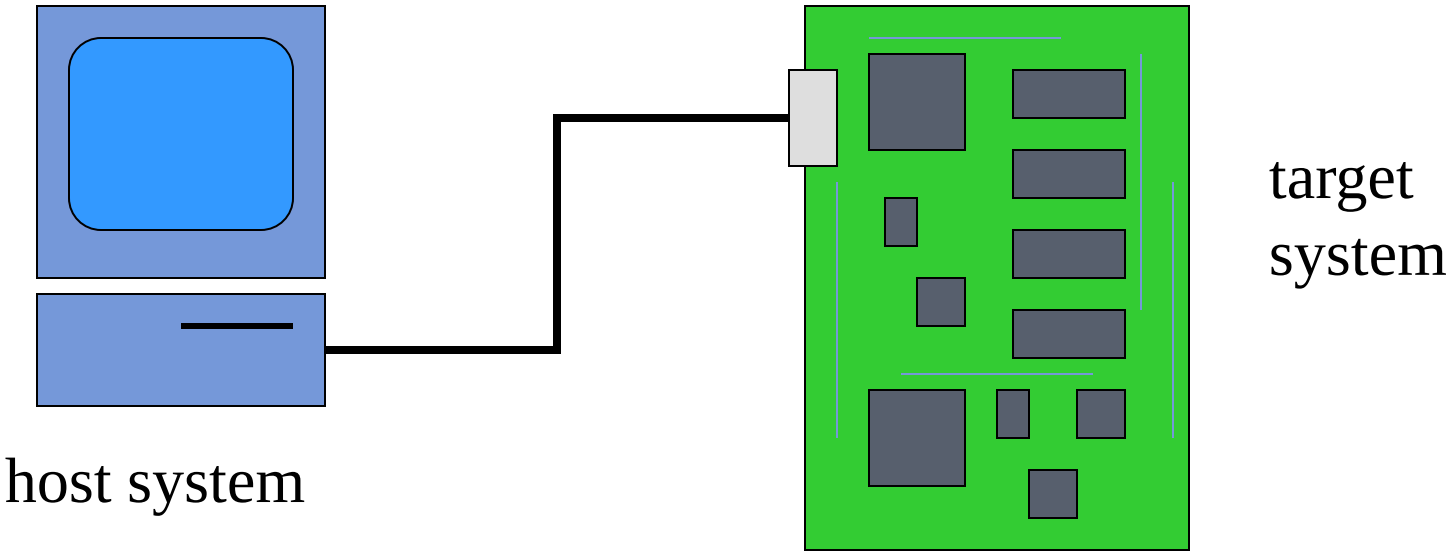
- software doesn't run without hardware;
- how much hardware you need can be largely determined by the software requirements:
  - Speed;
  - Memory size;
  - Interconnection bandwidth.

Software design techniques

- Want to develop as much code as possible on a standard platform:
  - friendlier programming environment;
  - easier debugging.
- May need to devise software stubs to allow testing of software elements without the full hardware/software platform.

Host/target design

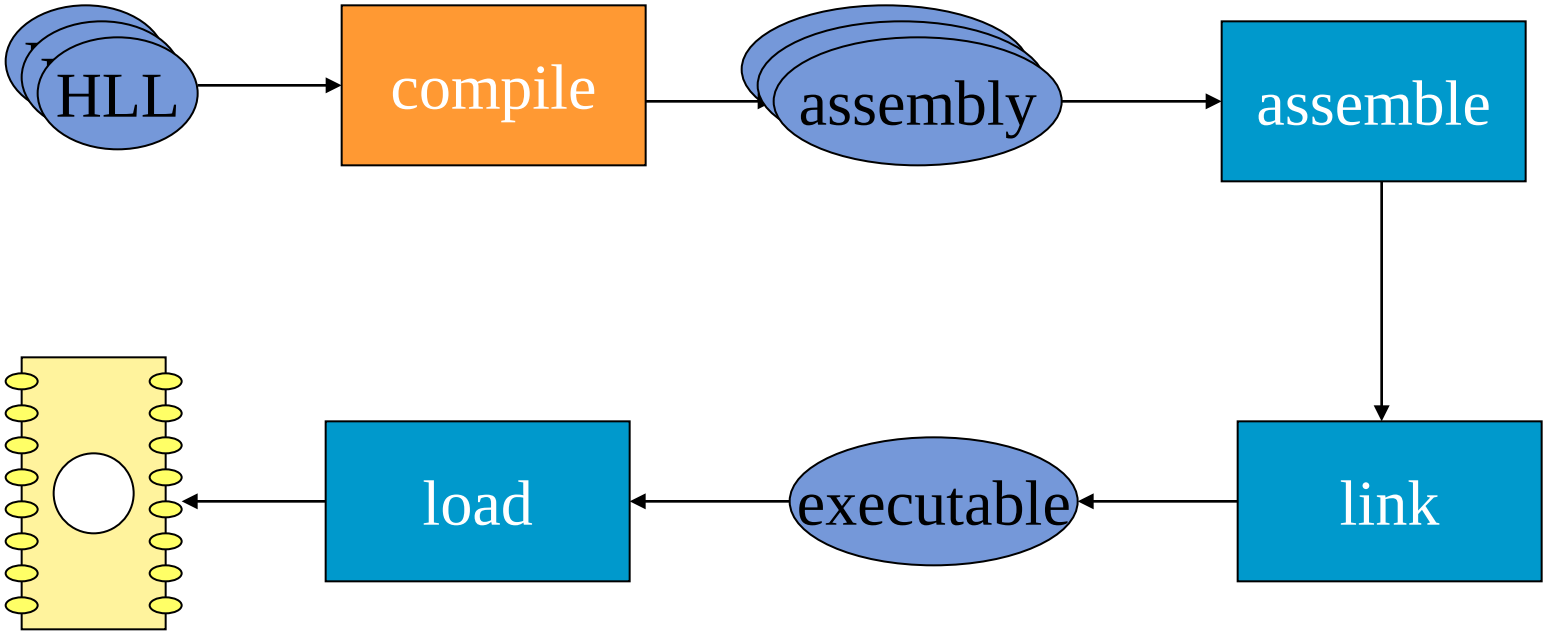
- Use a host system to prepare software for target system:



Cross-platform development environment

- The embedded computing system is usually tightly resource constrained.
- A PC or workstation is commonly used for development purpose
  - Cross compiler:
    - compiles code on host for target system.
  - Cross debugger:
    - displays target state, allows target system to be controlled.

Embedded Software Compilation

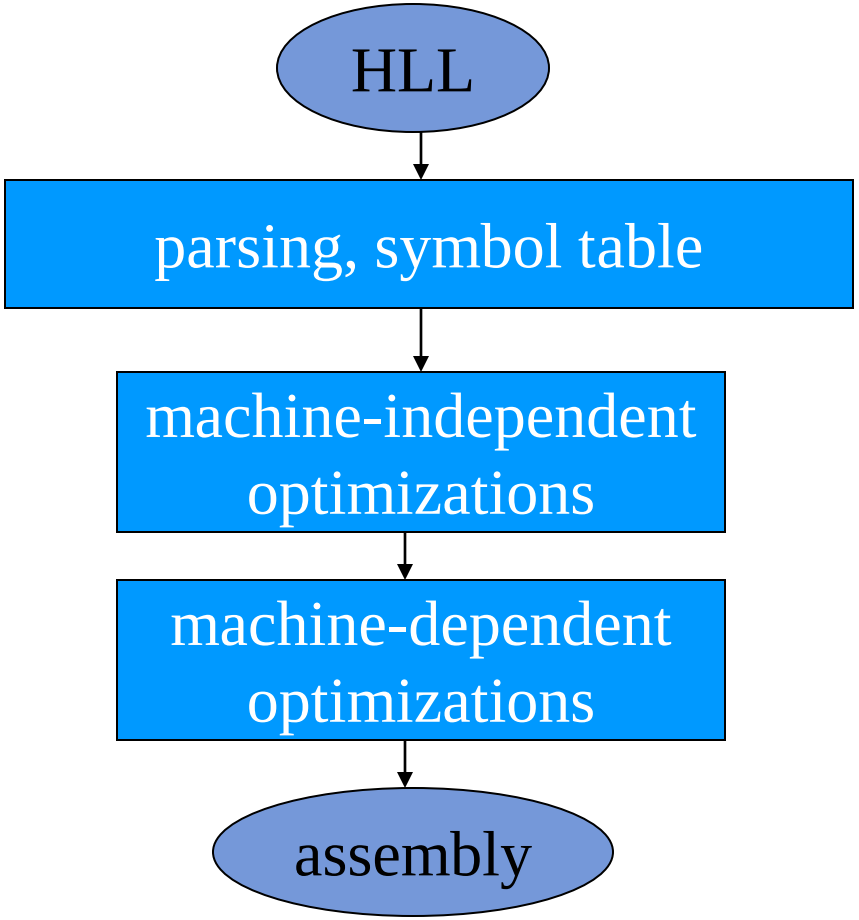


FIU
Florida International University

The Compiler

- Compilation = translation + optimization
- Compiler determines quality of code:
  - use of CPU resources;
  - memory access scheduling;
  - code size.

Basic compilation phases



FIU
Florida International University

Models of programs

- Source code is not a good representation for programs:
  - clumsy;
  - leaves much information implicit.
- Compilers derive intermediate representations to manipulate and optimize the program.
  - Data flow graph
  - Control data flow graph

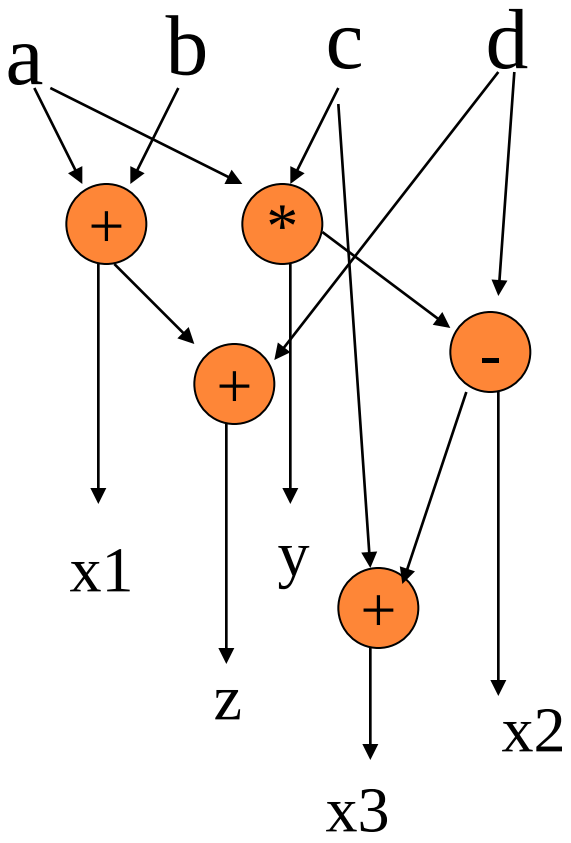
Data Flow Graph

- Definition
  - A directed graph that shows the data dependencies between a number of functions
    - Nodes
      - Representing operation
      - Each node having input/output data ports
    - Arces:
      - connections between the output ports and input ports

Data flow graph construction

single-assignment form:

```
x1 <= a + b;
y <= a * c;
z <= x1 + d;
x2 <= y - d;
x3 <= x2 + c;
```



FIU
Florida International University

Control-data flow graph

- CDFG: represents control and data.
- Uses data flow graphs as components.
- Two types of nodes:
  - decision;
  - data flow.

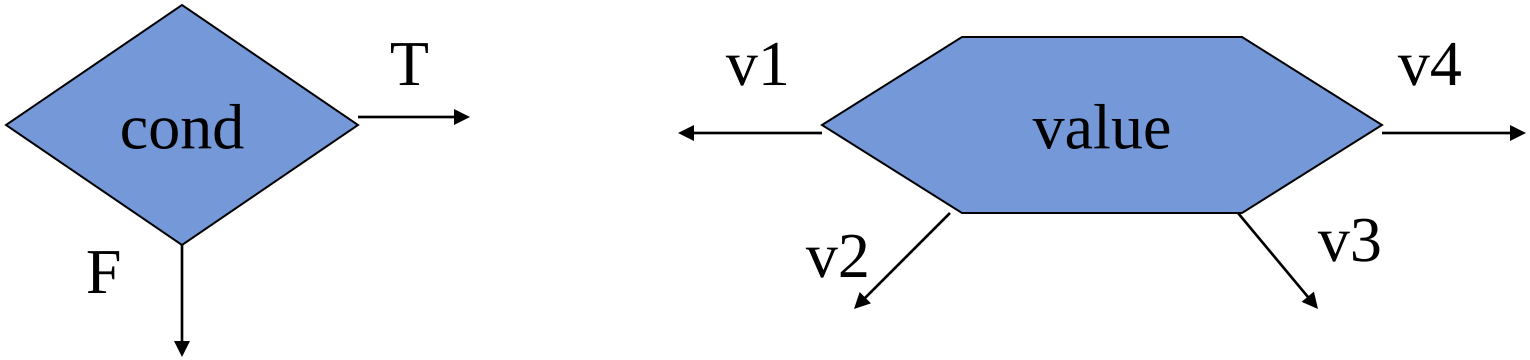
Data flow node

Encapsulates a data flow graph:

Write operations in basic block form for simplicity.

```
x = a + b;
y = c + d
```

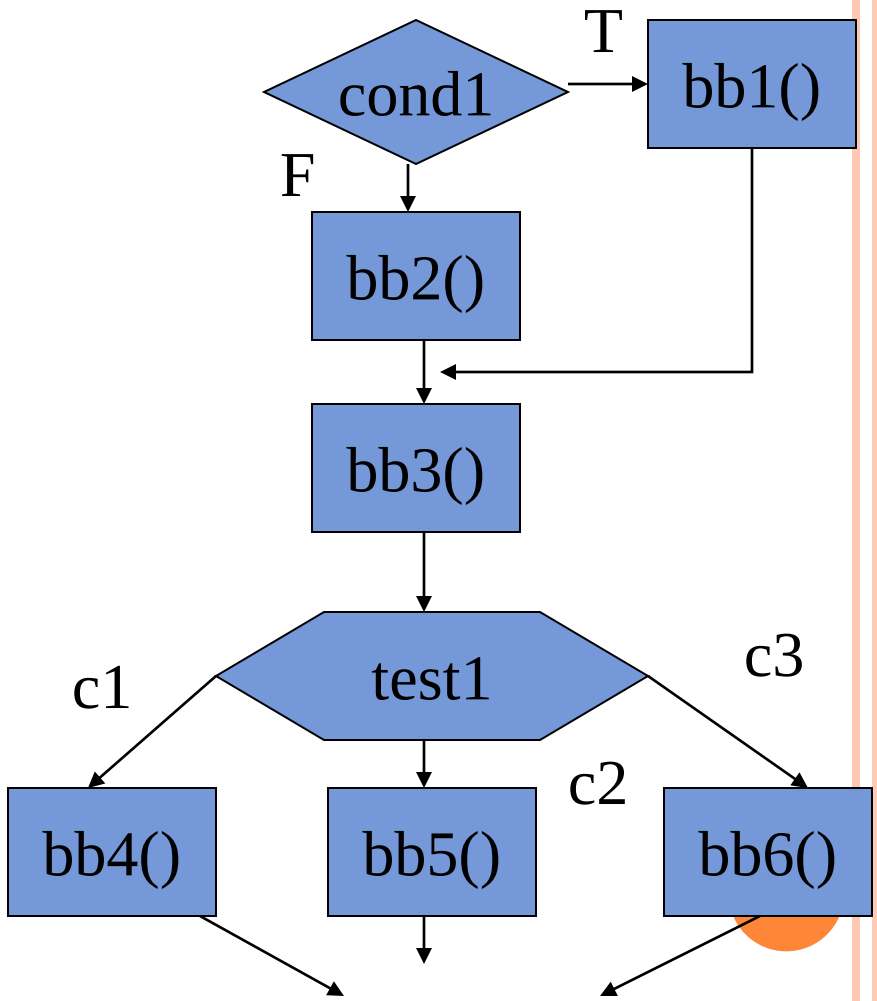
CONTROL



Equivalent forms

CDFG EXAMPLE

```
if (cond1) bb1();
 else bb2();
bb3();
switch (test1) {
    case c1: bb4(); break;
    case c2: bb5(); break;
    case c3: bb6(); break;
}
```

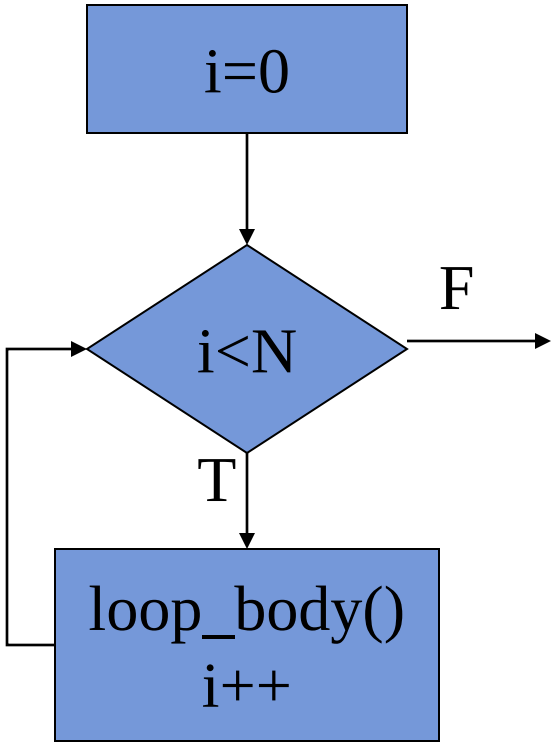


FOR LOOP

```
for (i=0; i<N; i++)
    loop_body();
```

for loop

```
i=0;
while (i<N) {
    loop_body(); i++;
    }
```



FIU
Florida International University

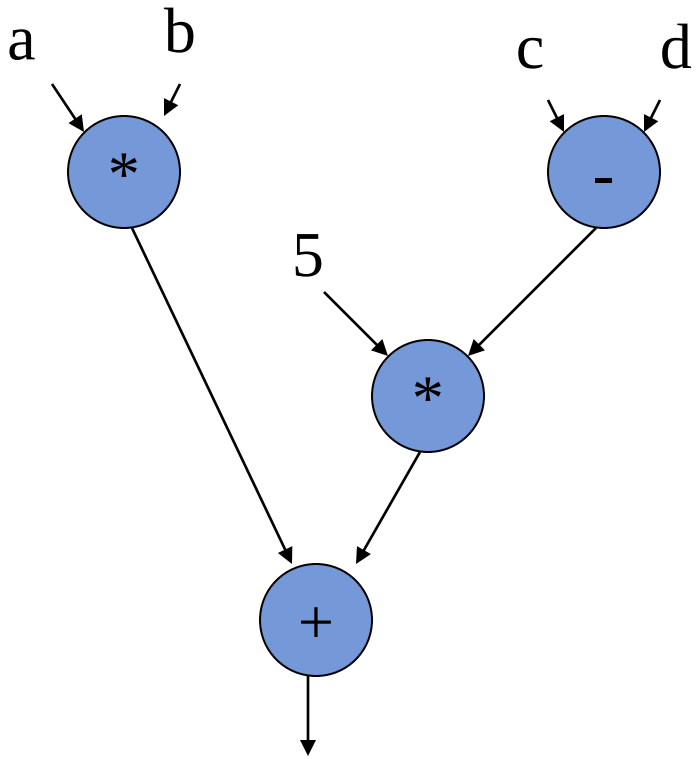
Translation and optimization

- Source code is translated into intermediate form such as CDFG.
- CDFG is transformed/optimized.
- CDFG is translated into instructions with optimization decisions.
- Instructions are further optimized.

Arithmetic expressions

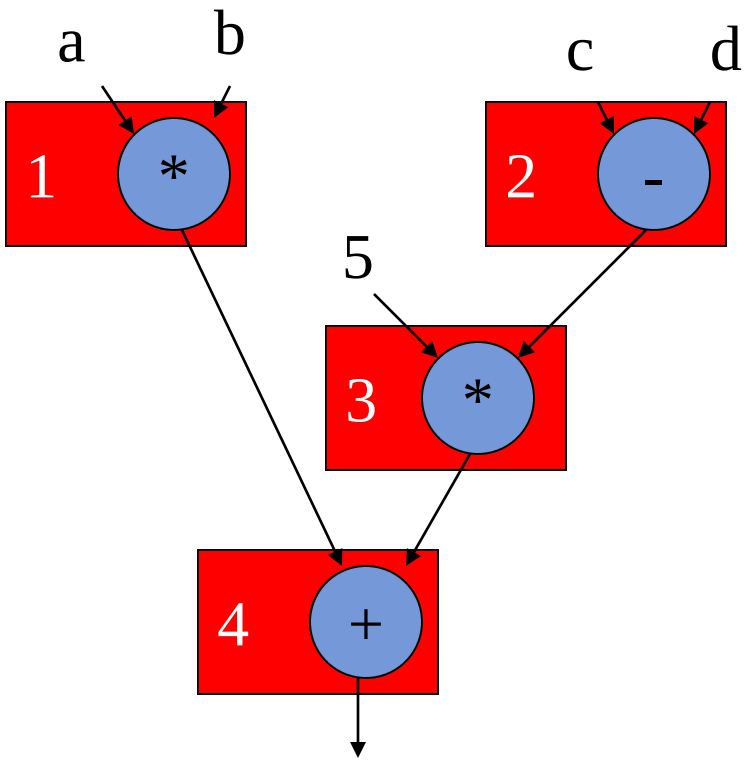
a*b + 5*(c-d)

expression



DFG

Arithmetic expressions, cont'd.



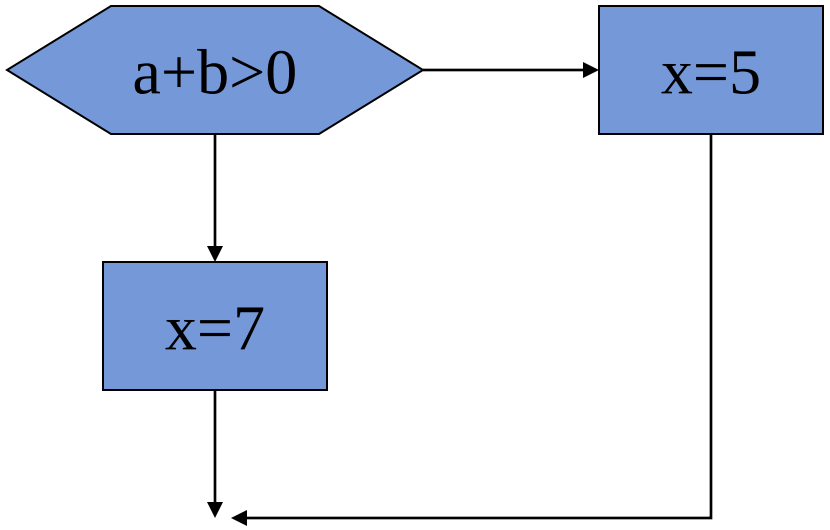
DFG

```
ADR r4,a
MOV r1,[r4]
ADR r4,b
MOV r2,[r4]
MUL r3,r1,r2
ADR r4,c
MOV r1,[r4]
ADR r4,d
MOV r5,[r4]
SUB r6,r4,r5
MUL r7,r6,#5
ADD r8,r7,r3
```

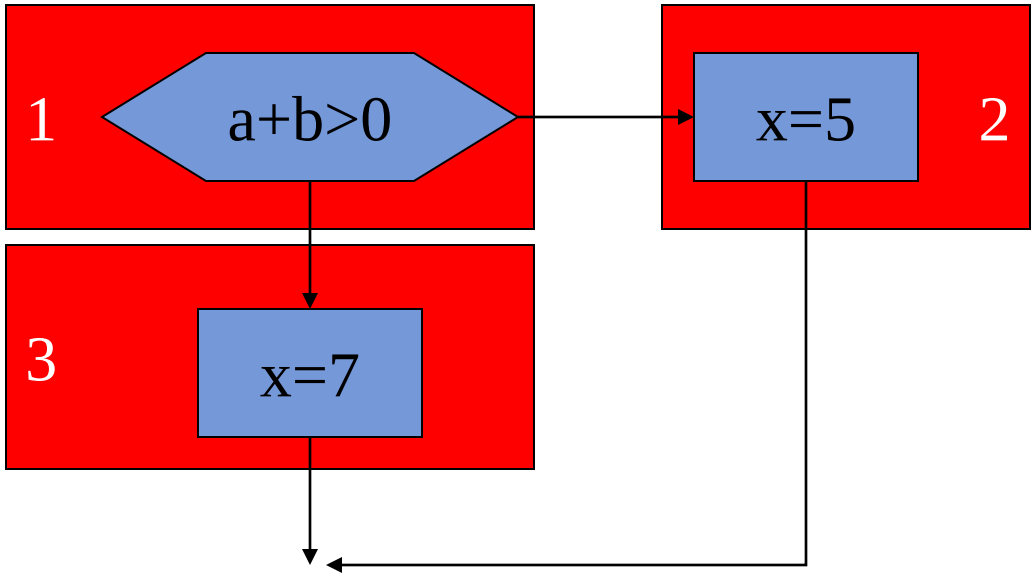
code

Control code generation

```
if (a+b > 0)
    x = 5;
else
    x = 7;
```



Control code generation, cont'd.




```
ADR r5,a
LDR r1,[r5]
ADR r5,b
LDR r2,b
ADD r3,r1,r2
BLE label3
LDR r3,#5
ADR r5,x
STR r3,[r5]

B stmtent
label3 LDR r3,#7
ADR r5,x
STR r3,[r5]
stmtent ...
```

FIU
Florida International University

Optimizations

- Machine independent
  - Expression simplification, Loop optimization
- Machine dependent
  - Register allocation
  - Instruction scheduling
  - Instruction selection

FIU
Florida International University

Expression simplification

- Constant folding:
  - 8+1 = 9
- Algebraic:
  - a*b + a*c = a*(b+c)
- Strength reduction:
  - a*2 = a<<1
- Common sub-expression reduction
  - x = (a+b*c) * d; y = (a+b*c)/d;

➔ t = (a+b*c); x = t * d; y = t/d;

FIU
Florida International University

Register allocation

- Goals:
  - choose register to hold each variable;
  - determine lifespan of variable in the register.
  - reduce *memory spills*
    - *Memory spills:* temporary data has to be stored in memory due to register shortage

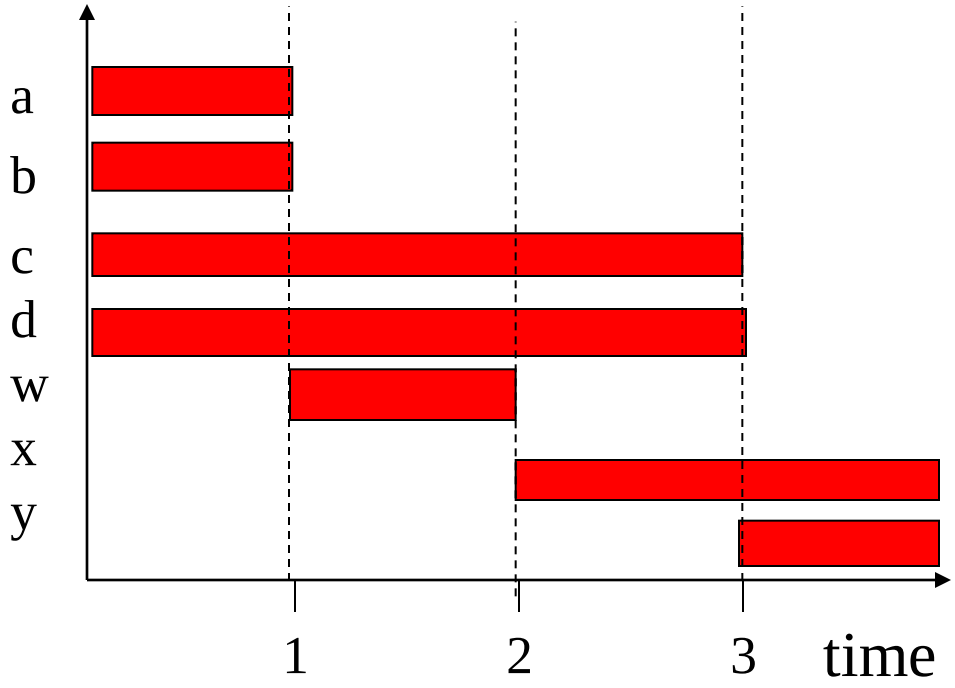
Register lifetime graph

w = a + b;

x = c + w;

y = c + d;

(assume x, y are the output variables for later use)



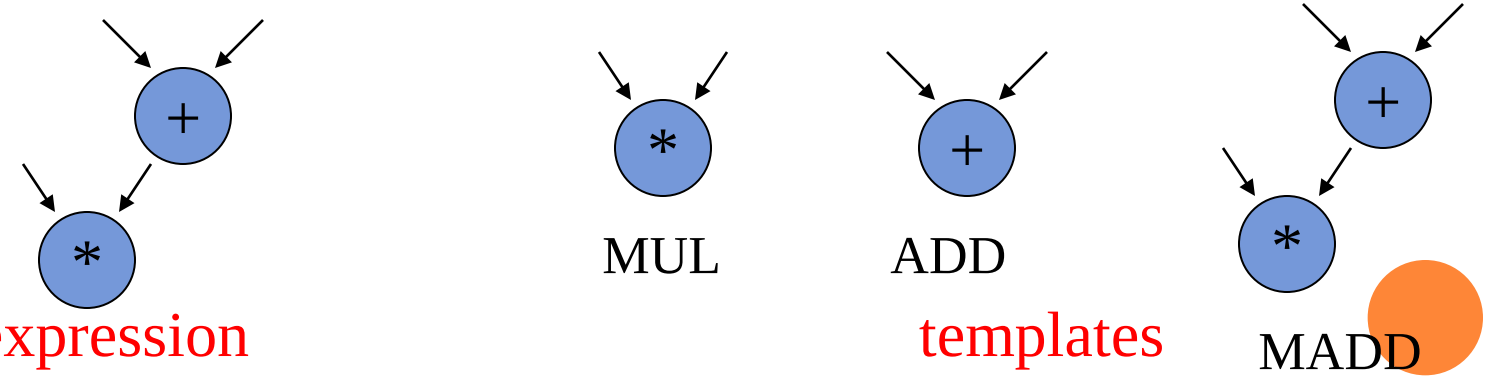
FIU
Florida International University

Instruction scheduling

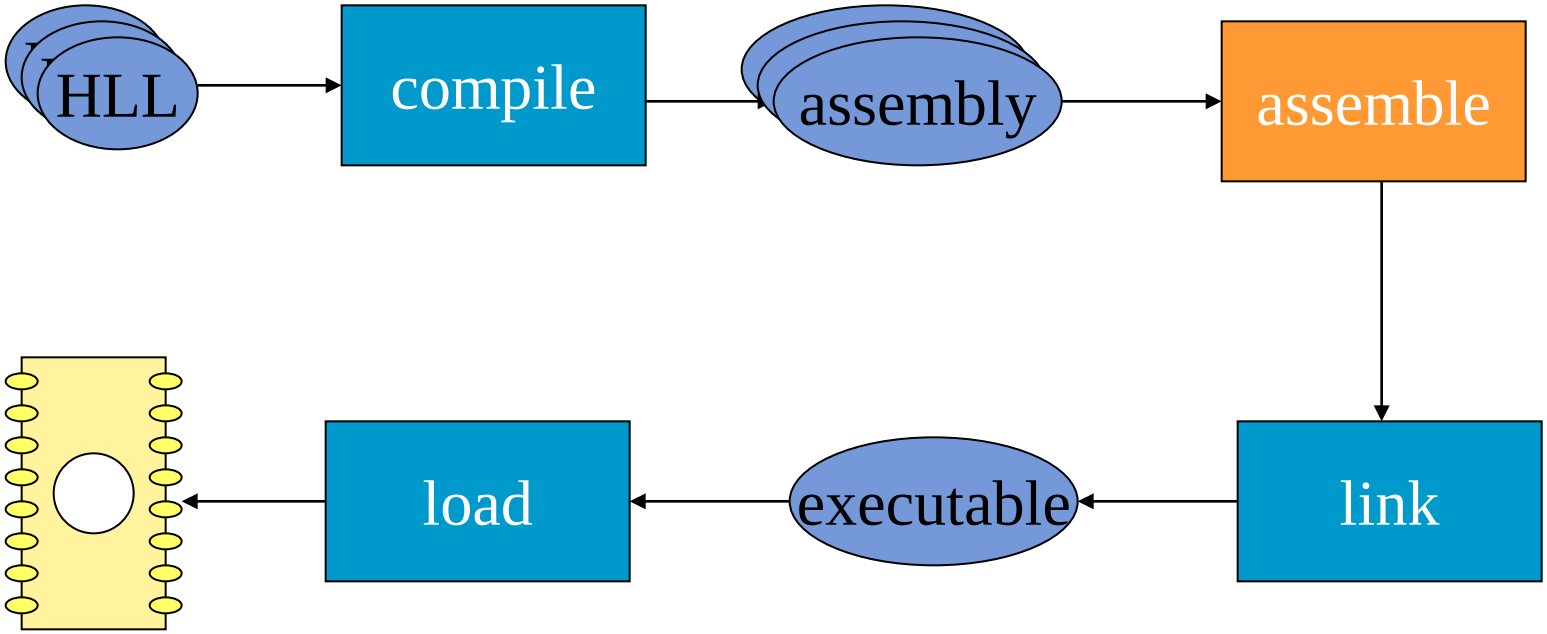
- Non-pipelined machines do not need instruction scheduling: any order of instructions that satisfies data dependencies runs equally fast.
- In pipelined machines, execution time of one instruction depends on the nearby instructions: opcode, operands.

Instruction selection

- May be several ways to implement an operation or sequence of operations.
- Represent operations as graphs, match possible instruction sequences onto graph.



Embedded Software Compilation



Assemblers

- Major tasks:
  - generate binary for symbolic instructions;
  - translate labels into addresses;
  - handle pseudo-ops (data, etc.).
- Generally one-to-one translation.
- Assembly labels:

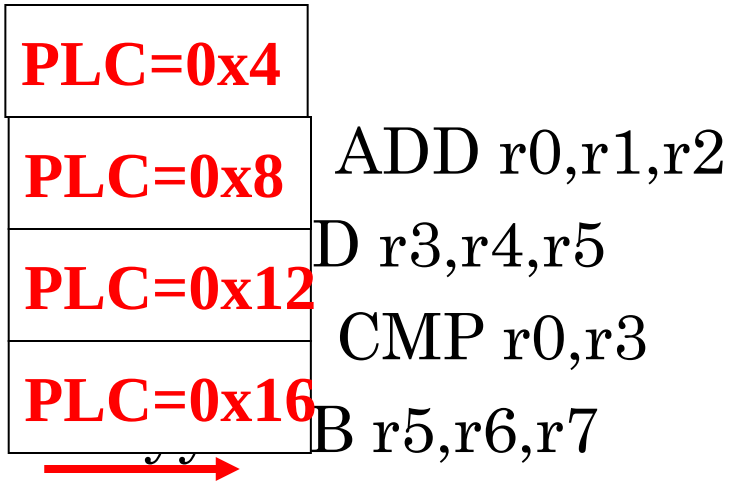
```
            ORG 100
label1      ADR r4,c
```

FIU
Florida International University

Two-pass assembly

- Pass 1:
  - generate symbol table
- Pass 2:
  - generate binary instructions

Symbol table example



Symbol Table

xx 0x8

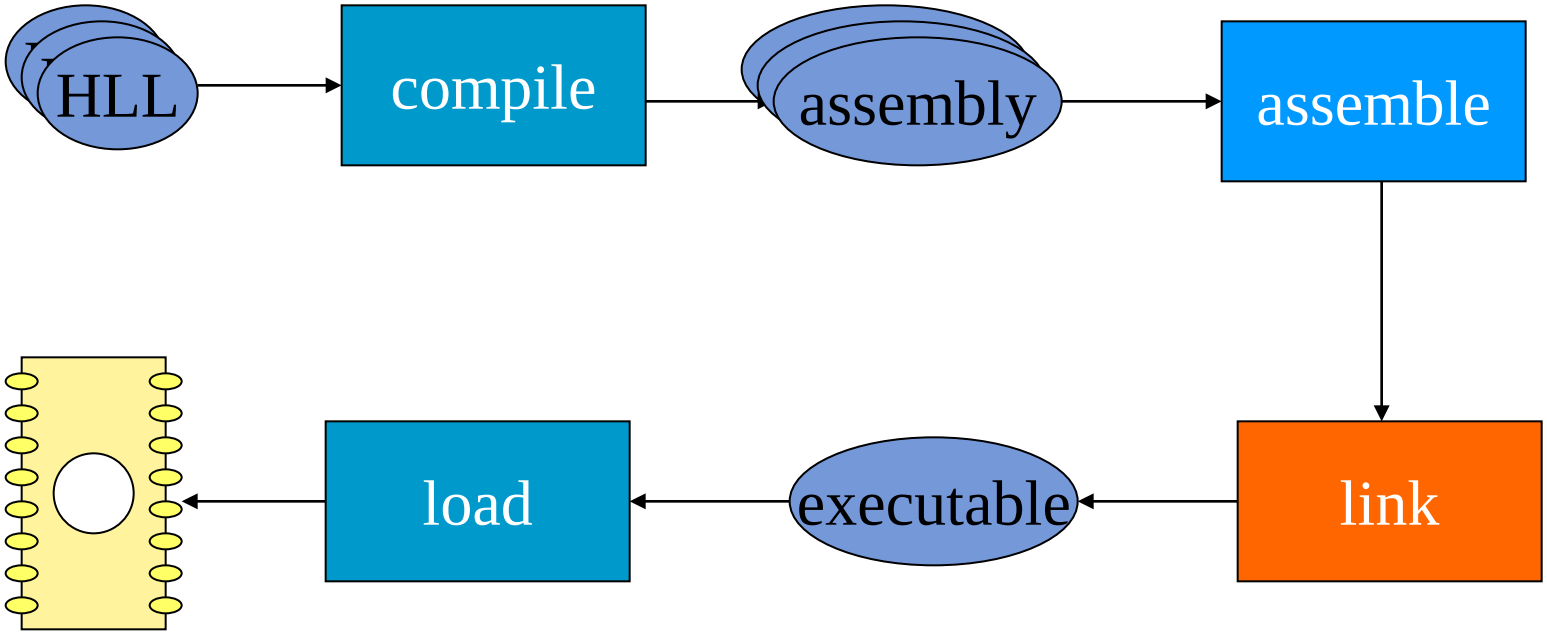
yy 0x16

FIU
Florida International University

Pseudo-operations

- Pseudo-ops do not generate instructions:
  - ORG sets program location.
  - EQU generates symbol table entry without advancing PLC.
  - Data statements define data blocks.

Embedded Software Compilation

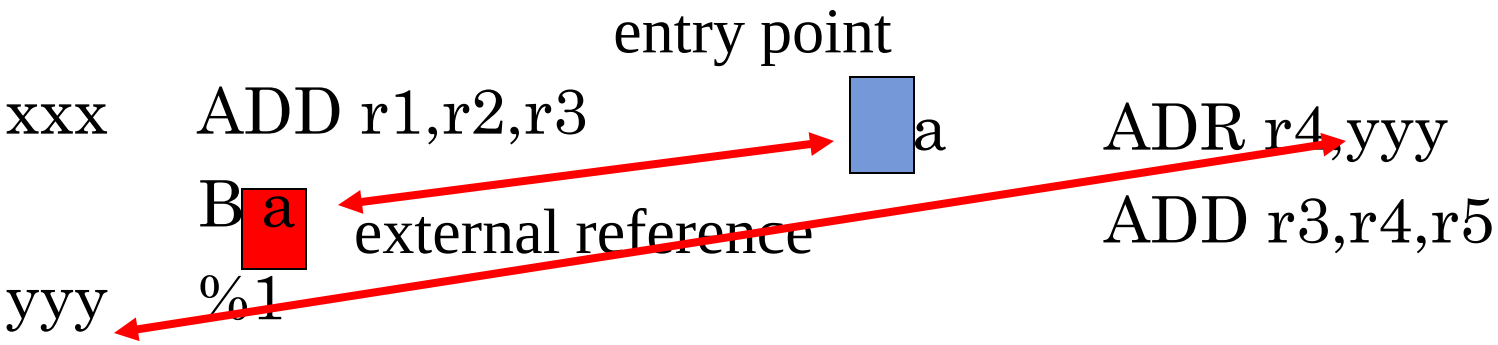


FIU
Florida International University

Linker

- Combines several object modules into a single executable module.
- Jobs:
  - put modules in order;
  - resolve labels across modules.

Externals and entry points

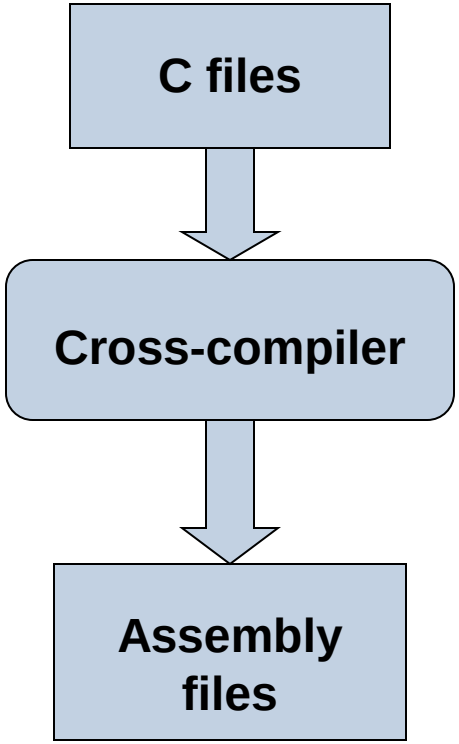


Dynamic linking

- Some operating systems link modules dynamically at run time:
  - shares one copy of library among all executing programs;
  - allows programs to be updated with new versions of libraries.

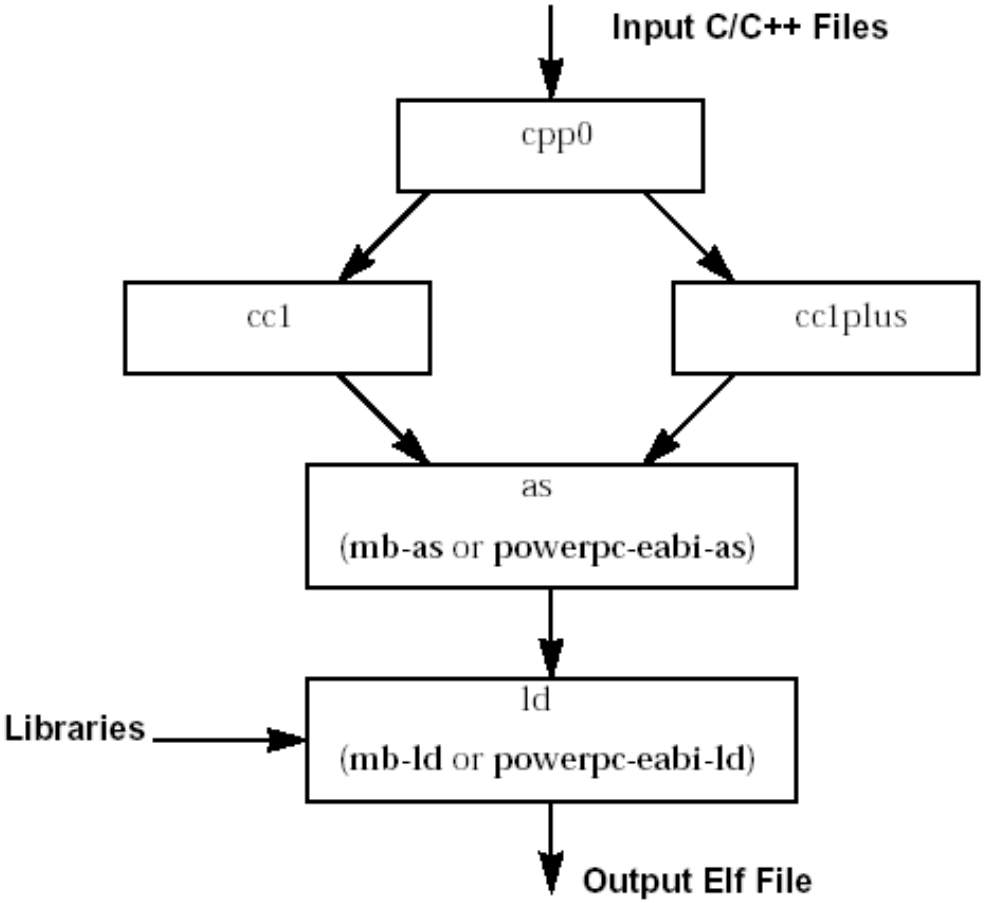
GNU Tools: GCC

- GCC translates C source code into assembly language
- GCC also functions as the user interface to the GNU assembler and to the GNU linker, calling the assembler and the linker with the appropriate parameters
- Supported cross-compilers:
  - PowerPC™ processor compiler
    - GNU GCC (powerpc-eabi-gcc)
  - MicroBlaze™ processor compiler
    - GNU GCC (mb-gcc)



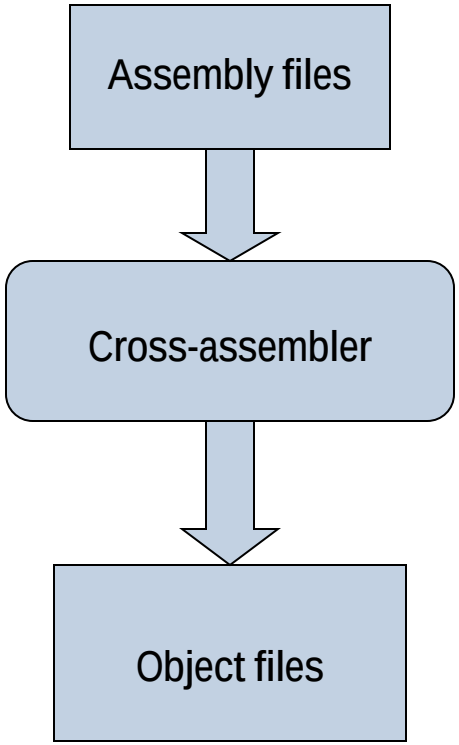
GNU Tools

- Calls four different executables
  - Preprocessor (cpp0)
  - Language specific c-compiler
    - cc1 C-programming language
    - cc1plus C++ language
  - Assembler
    - mb-as (MicroBlaze™ processor)
    - powerpc-eabi-as (PowerPC™ processor)
  - Linker and loader
    - mb-ld (MicroBlaze processor)
    - powerpc-eabi-ld (PowerPC processor)



GNU Tools: AS

- Input: Assembly language files
  - File extension: **.s**
- Output: Object code
  - File extension: **.o**
  - Contains
    - Assembled piece of code
    - Constant data
    - External references
    - Debugging information
- Typically, the compiler automatically calls the assembler



Object File Sections

- What is an object file?
  - An object file is an assembled piece of code
    - Machine language:
      li r31,0 = 0x3BE0 0000
  - Constant data
  - There may be references to external objects that are defined elsewhere
  - This file may contain debugging information

Object File Sections

.text	Text section
.rodata	Read-only data section
.sdata2	Small read-only data section (less than eight bytes)
.data	Read-write data section
.sdata	Small read-write data section
.sbss	Small uninitialized data section
.bss	Uninitialized data section

Object File Sections

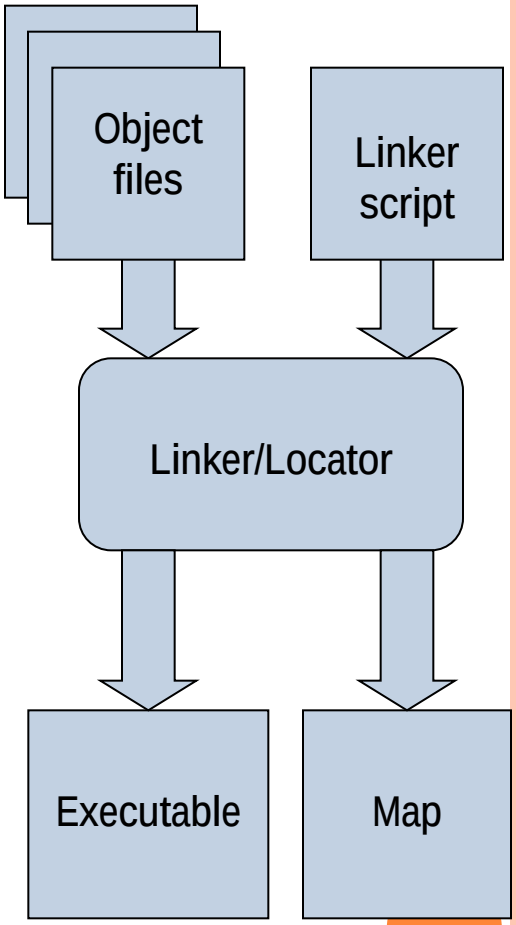
.init	Language initialization code
.fini	Language cleanup code
.ctors	List of functions to be invoked at program startup
.dtors	List of functions to be invoked at program end
.got2	Pointers to program data
.got	Pointers to program data
.eh_frame	Frame unwind information for exception handling

Sections Example

```
int ram_data[10] = {0,1,2,3,4,5,6,7,8,9};       /* DATA */

const int rom_data[10] = {9,8,7,6,5,4,3,2,1};   /* RODATA */

int I;   /* BSS */

main(){

...
I = I + 10;  /* TEXT */
...

}
```

GNU Tools: LD

- Linker
- Inputs:
  - Several object files
  - Archived object files (library)
  - Linker script: how different sections of input should be put in output files
- Outputs:
  - Executable image (.ELF)
    - Executable and linking format
      - a common standard file format for executables, object code, shared libraries, etc.
  - Mapfile
    - the memory layout

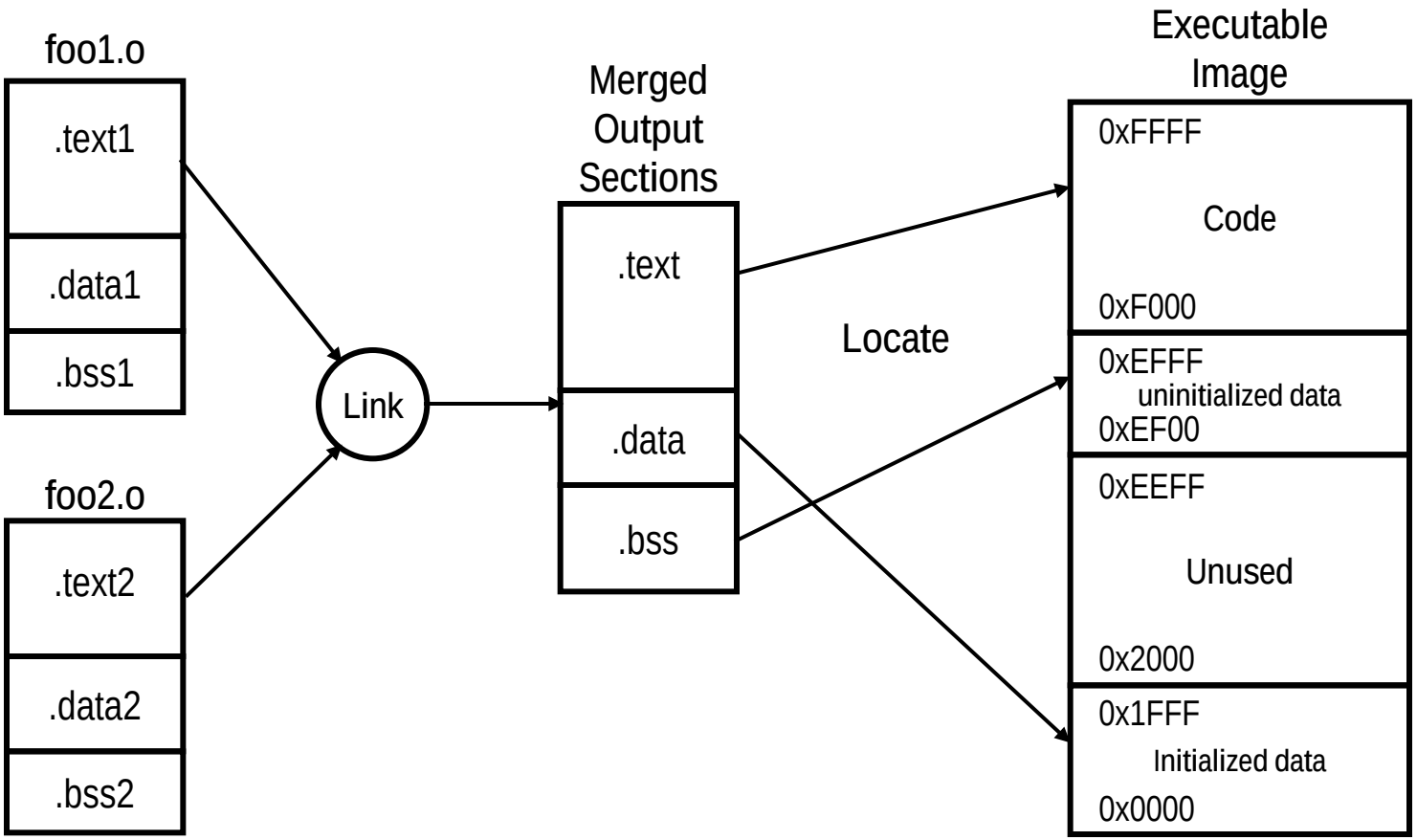


FIU
Florida International University

Linker Scripts

- Linker scripts
  - Control the linking process
  - Map the code and data to a specified memory space
  - Set the entry point to the executable
  - Reserve space for the stack
- Required if the design contains a discontinuous memory space

Linker and Locator Flows



PowerPC Processor Script Example

```
STACKSIZE = 4k;
MEMORY
{
  ddr   : ORIGIN = 0x00000000, LENGTH = 32m
  sram  : ORIGIN = 0x10000000, LENGTH = 2m
  flash : ORIGIN = 0x18000000, LENGTH = 32m
  bram  : ORIGIN = 0xffff8000, LENGTH = 32k - 4
  boot  : ORIGIN = 0xfffffffc, LENGTH = 4
}
SECTIONS
{
  .text    : { *(.text) } > bram
  .boot    : { *(.boot) } > boot
  .data    : { *(.data) *(.got2) *(.rodata) *(.fixup)} > bram
  .bss     : { *(.bss) } > bram
  __bss_start = ADDR(.bss);
  __bss_end   = ADDR(.bss) + SIZEOF(.bss);
}
```

Binutils: Binary Utilities

- AR Archiver
  - Create, modify, and extract from libraries
  - Used in EDK to combine the object files of the Board Support Package (BSP) in a library
  - Used in EDK to extract object files from different libraries
- OBJDUMP
  - Display information from object files and executables
    - Header information, memory map
    - Data
    - Disassemble code
  - GNU executables
    - powerpc-eabi-objdump
    - mb-objdump

FIU
Florida International University

Summary

- Cross-platform design environment
- Cross-platform Compilation
  - Compiling and optimization
  - Assembling and Linking
- GNU Tools