assembly ⇒ assembler ⇒ merl

merl ⇒ loader ⇒ loads + relocates

assembly ⇒ Assembler ⇒ MERL

MERL

Note: MERL files are executable as is (if loaded at address 0)

Header	1. jumps next 2 words 2. endModule 3. endCode
Assembled Code	
Footer	**REL** **[word 1] :** format specifier = 1 **[word 2] :** address of word to relocate

Example 1					
ASSEMBLY		**ML**	**MERL**		
			beq $0, $0, 2 .word endModule .word endCode	0x00 0x04 0x08	0x10000002 0x0000003c 0x0000002c
lis $3 .word 0xabc lis $1 .word A jr $1 B: jr $31 A: beq $0, $0, B .word B	0x00 0x04 0x08 0x0C 0x10 0x14 0x18 0x1C	0x00001814 0x00000abc 0x00000814 0x00000018 0x00200008 0x03e00008 0x1000fffe 0x00000014	lis $3 .word 0xabc lis $1 reloc1: .word A jr $1 B: jr $31 A: beq $0, $0, B reloc2: .word B	0x0c 0x10 0x14 0x18 0x1c 0x20 0x24 0x28	0x00001814 0x00000abc 0x00000814 0x00000024 0x00200008 0x03e00008 0x1000fffe 0x00000020
			endCode: .word 1 .word reloc1 .word 1 .word reloc2	 0x2c 0x30 0x34 0x38	 0x00000001 0x00000018 0x00000001 0x00000028

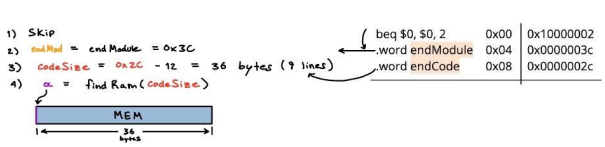
	endModule:	

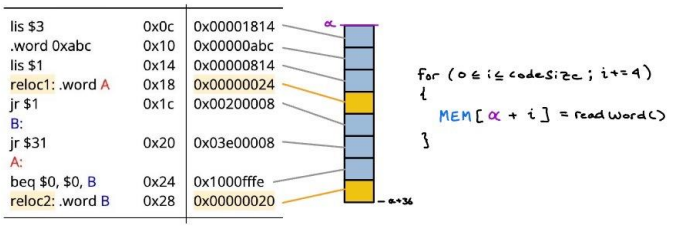
Example 2					
ASSEMBLY		**ML**	**MERL**		
			beq $0, $0, 2 .word endModule .word endCode	0x00 0x04 0x08	0x10000002 0x0000003c 0x0000002c
lis $1 .word 0x1000 lis $2 .word A A : jr $2 beq $0 , $1 , B B : jr $31	0x00 0x04 0x08 0x0C 0x10 0x14 0x18	0x . . . 0x . . . 0x . . . 0x00000010 0x . . . 0x . . . 0x . . .	lis $1 .word 0x1000 lis $2 reloc1: .word A A : jr $2 beq $0 , $1 , B B : jr $31	0x0c 0x10 0x14 0x18 0x1c 0x20 0x24	0x . . . 0x . . . 0x . . . 0x0000001c 0x . . . 0x . . . 0x . . .
			endCode: .word 1 .word reloc1 endModule:	 0x28 0x2c 0x30	 0x00000001 0x00000018

Loader: MERL ⇒ program in memory

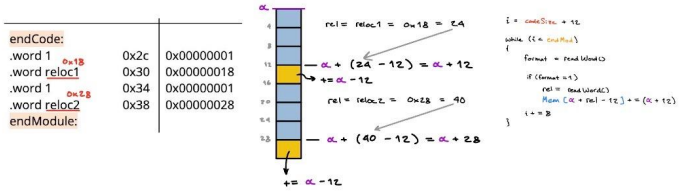
1. Decides **how much memory** is required
2. Loads into memory at **α**
3. **relocate** from **α**
4. return **α** to **OS**

Algorithm

read_word ()	
endMod = read_word()	
codeSize = read_word() - 12	
LOADING	

α = findFreeRAM(codeSize)
 	
for(word **i** < codeSize, i += 4)
{
  Load word **i** ⇒ MEM[α + i]
}
	

RELOCATE	

i = codeSize + 12
 	
while (i < endMod)
{
  if (format == 1)
  rel = read_word()
  MEM [α + *rel* - 12] += (α - 12)
}
	

MERL files ⇒ Linker ⇒ linked.MERL

1. For every file: file ⇒ assembler ⇒ MERL
2. **linking algorithm**
3. loader

Another Assembler Update

External Symbol Reference (ESR)	**External Symbol Definitions (ESD)**
[label can't be resolved] **+** [".import label" exists]	[file provides label] + [export label exists]

File	**Merl in Assembly**		**MERL Output**
	beq $0 , $0 , 2 .word endModule .word endCode	0x00 0x04 0x08	0x10000002 0x00000034 0x00000018
.import *proc* lis $1 .word proc jalr $1	lis $1 use1: .word 0 ; proc placeholder jalr $1	0x0c 0x10 0x14	0x00000814 0x00000000 0x00200009

	endCode: .word 0x11 ; **ESR** .word use1 ; **proc** address ... endModule:	0x18 0x1c 0x20 0x24 0x28 0x2c 0x30	0x00000011 0x00000010 0x00000004 0x00000070 0x00000072 0x0000006f 0x00000063

proc

- placeholder output for this word (0) at this location
- replaced by the linker once the actual address of *proc* is known

File	Merl in Assembly		MERL Output
	beq $0 , $0 , 2 .word endModule .word endCode	0x00 0x04 0x08	0x10000002 0x0000002c 0x00000010
.export proc proc: jr $31	proc: jr $31	0x0c	0x03e00008
	endCode: .word 0x05 ;**format code** for ESD .word proc ;***proc*** address ... endModule:	0x10 0x14 0x18 0x1c 0x20 0x24 0x28	0x00000005 0x0000000c 0x00000004 0x00000070 0x00000072 0x0000006f 0x00000063

Linking Algorithm

Link MERL files: **m1** and **m2**

1	**Check for duplicate exports**
2	**m1.code + m2.code**

3	**Relocate m2.table**	**Update REL, ESD and ESR entries** 1. reloc_offset = end of m1.code - 12 2. add reloc_offset to the # second word
4	**Relocate m2.code REL's**	Add reloc_offset to each ".word label" in code
5	**Resolve imports for m1/m2**	1. ESR location = exported label address 2. ESR ⇒ REL entry
6	**Combine Updated Tables**	
7	**Compute Header**	1. beq $0, $0, 2 2. endModule = endCode + table.size 3. endCode = code.size + 12

Example