Tutorial 5

- Linking & Loading
- MERL

Linking & Loading

- Loader copies our MIPS programs into memory
  - ↳ We assume it always copies our progs into MEM address 0x00 ← Unrealistic!
  - ↳ The loader is provided by the operating system, OS controls what progs are run from disk
- Why is loading progs at 0x00 not realistic?
  - ↳ The OS is a prog that needs to also be loaded in MEM!
  - ↳ Modern OS's support multiple progs running at the same time

OS pseudocode:

```
; OS:
repeat:
    P = choose prog to run
    $3 = loader(P)      ; $3 = starting addr of P
    jalr $3
    beq  $0, $0, repeat
```

```
; Loader
startAddr = findFreeRAM(N)      ; N = # of MEM allocated
for (i=0; i < codeLen; i++){
        MEM[startAddr + 4*i] = P[i]   ; load in prog P
}
$30 = startAddr + N             ; Set stack address
return startAddr
```

start of a block of MEM big enough for P

- This all seems good but one problem, how do we handle `.word label` ?
  - ↳ need loader to relocate all labels with label + offset
  - ↳ hard to identify in binary

```
01101011001 0 ...
01001101000 ...
0000 111001 . -.
1100 1101000 ...
```

} Loader doesn't know what lines should be offset!

- Soln: have <u>the assembler</u> encode information on what locations need relocation (ie: where all `.word label` lines exist)
  - ↳ generate object code instead of pure binary
  - ↳ CS241: MIPS Executable Relocatable Linkable (MERL)

MERL

- 3 components

1) Header (3 words of info)
   - MERL Cookie : 0x10000002
     → MERL file identifier (≡ beq $0, $0, 2)
   - End of Module address
     → One word past the last word
   - End of Code address
     → End of code, start of footer

2) MIPS Code
   - Normal machine code prog
   - All labels shifted by the loader!
   - Header offset by 3 words (0x0c)

3) Footer
   - Contains 3 types of "entries"

   ↳ REL (Relocation):
   → specify MIPS code addresses that need relocation (ie: .word label)
   → Code (2 words):
   [0x00000001 ← REL code
   ... address of code to be relocated...]

   Used by Loader

   ↳ ESR (External Symbol Reference)
   ↳ ESD (External Symbol Definition)

   Used by Linker

Loader

eg: Convert the following assembly into MERL format

Address	Assembly	ML (in HEX)	
0x00	lis $3	0x00001814	
0x04	.word 0xabc	0x00000abc	
0x08	lis $1	0x00000814	
0x0c	.word A	0x00000018	; REL A
0x10	jr $1 B:	0x00200008	
0x14	jr $31 A:	0x03e00008	
0x18	beq $0, $0, B	0x1000fffe	
0x1c	.word B	0x00000014	; REL B

Address	MERL	MERL in Asm
0x00	0x10000002	beq $0, $0, 2
0x04	0x0000003c	.word endModule
0x08	0x0000002c	.word endCode
0x0c	0x00001814	lis $3
0x10	0x00000abc	.word 0xabc
0x14	0x00000814	lis $1
0x18	0x00000024 ← (1)	reloc1: .word A
0x1c	0x00200008	jr $1
0x20	0x03e00008	B: jr $31
0x24	0x1000ffff	A: beq $0, $0, B
0x28	0x00000020 ← (2)	reloc2: .word B
		endCode:
0x2c	0x00000001 ← REL	.word 1
0x30	0x00000018 ← .word A	.word reloc1
0x34	0x00000001 ← REL	.word 2
0x38	0x00000028 ← .word B	.word reloc2
0x3c		endModule:

Header

REL Calculations: MERL header offset

(1) 0x00000018 + 0x0000000c = 0x00000024

(2) 0x00000014 + 0x0000000c = 0x00000020

Note: When loading a MERL file to address x, we do not load the header

↳ ∴ all .word label ⇒ startCode + relAddr − 0x0c

eg: load the previous MERL file to address = 0x1240

MERL	Address	Loaded Code	
Not loaded: 0x10000002	0x1240	0x00001814	lis $3
Not loaded: 0x0000003c	0x1244	0x0000abc	.word 0xabc
Not loaded: 0x0000002c	0x1248	0x00000814	lis $1
0x00001814	0x124c	0x00001258	.word A
0x00000abc	0x1250	0x00200008	jr $1
0x00000814	0x1254	0x03e00008	B: jr $31
0x00000024	0x1258	0x1000fffe	A: beq, $0, $0, B
0x00200008 →	0x125c	0x00001254	.word B
0x03e00008			
0x1000fffe			
0x00000020 →			
0x00000001			
0x00000018			
0x00000001			
0x00000028			

0x00000018 ← REL .word A = 0x24 + 0x1240 − 0x0c

= 0x1258 ← A: address!

0x00000028 ← REL .word B = 0x20 + 0x1240 − 0x0c

= 0x1254 ← B: address!

Linking

- Goal: load 2^+ MERL files together
  - ↳ Need to relocate labels after combination
  - ↳ Need to handle cross-file label references
- ESR (External Symbol Reference)
  - ↳ generated if:
    1) A label is referenced by .word label and
    2) Your file contains .import label
  - ↳ Stores address of the .word label in MERL file
  - ↳ Code (3 + n words)

3 words:
- 0x00000011 ← ESR code
- ... address of label reference ...
- ... length n of label name ...

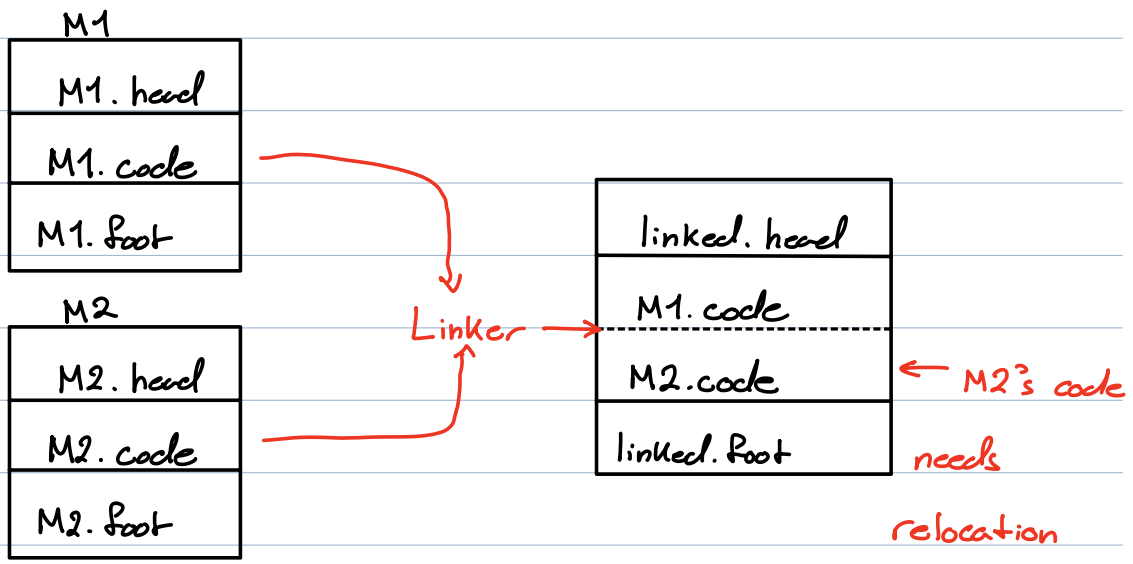
n words:
- ... one word per ASCII char in label name ...
- ⋮

- ESD (External Symbol Definition)
  - ↳ generated when
    1) label is defined with label: and
    2) label is exported with .export label
  - ↳ Stores address of label: defn in MERL file
  - ↳ Code (3 + n words)

3 words:
- 0x00000005 ← ESD code
- ... address of label defn ...
- ... length n of label name ...

n words:
- ... one word per ASCII char in label name ...

- Linking Algorithm (to link MERL files M1 & M2):
  1. Check for duplicate exports
     - ensure M1 & M2 don't export the same labels
  2. Combine MIPS code segments
     - Order of linking matters! linking M1 before M2 will cause M1's code to appear above M2



  3. Relocate M2.foot by updating the addresses of all REL, ESD & ESR addresses
     - ∀ ESD, ESR, REL adresses in M2.foot, add M1's code end address & subtract 0x0c (linked.head offset of 3 words)
  4. Relocate M2.code using REL entries
     - update each REL address from M2.foot in M2.code by the relocation offset from (3)

⑤ Resolve M1 imports

↳ ∀ ESR ∈ M1.bool, check if M2.bool has a corresponding ESD

↳ to resolve the import:

① replace the word of the label in M1.code with the corresponding address of the exported label

② Change the ESR entry into a REL entry ∵ the address of the import is now relocatable!

⑥ Resolve imports for M2

↳ repeat task ⑤ with M1 & M2's roles swapped

⑦ Create linked.bool

↳ Concatenate all ESD & REL entries (& all unresolved ESR entries) from M1.bool & M2.bool

⑧ Create linked.head

↳ Same MERL header syntax, endCode & endModule offsets need to consider the combined M1 & M2

eg: Show the result of linking one.asm (M1) & two.asm (M2)

one.asm

```
.import proc   ← ESR
lis $1
.word proc
jalr $1
```

two.asm

```
.export proc   ← ESD
proc:
jr $31
```

One.asm MERL

Address	Assembly	MERL Hex
0x00	beq $0, $0, 2	0x10000002
0x04	.word endModule	0x00000034
0x08	.word endCode	0x00000018
0x0c	lis $1	0x00000814
0x10	use1: .word 0	0x00000000
0x14	jalr $1	0x00200009
	endCode:	
0x18	.word 0x11	0x00000011
0x1c	.word use1	0x00000010
0x20	.word 4	0x00000004
0x24	.word 112	0x00000070
0x28	.word 114	0x00000072
0x2c	.word 111	0x0000006f
0x30	.word 99	0x00000063
0x34	endModule:	

header

code start

← temp value

← ESR code

← Addr of import

← len "proc"

← "p"

← "r"

← "o"

← "c"

two.asm MERL

Address	Assembly	MERL Hex
0x00	beq $0, $0, 2	0x10000002
0x04	.word endModule	0x00000010
0x08	.word endCode	0x0000002c
	proc:	
0x0c	jr $31	
	endCode:	
0x10	.word 0x05 ← ESD code	0x00000005
0x14	.word proc ← Addr of export	0x0000000c
0x18	.word 4 ← len "proc"	0x00000004
0x1c	.word 112 ← "p"	0x00000070
0x20	.word 114 ← "r"	0x00000072
0x24	.word 111 ← "o"	0x0000006f
0x28	.word 99 ← "c"	0x00000063
0x2c	endModule:	

header

code start

footer start

Result of Linking:

Address	Assembly	MERL Hex
0x00	beq $0, $0, 2	0x10000002
0x04	.word endModule	0x00000040
0x08	.word endCode	0x0000001c
0x0c	lis $1	0x00000814
0x10	use1: .word 0x18	0x00000018
0x14	jalr $1	0x00200009
	proc:	
0x18	jr $31	0x03e00008
	endCode:	
0x1c	.word 0x05 ← ESD	0x00000005
0x20	.word proc	0x00000018
0x24	.word 4 len("proc")	0x00000004
0x28	.word 112 "p"	0x00000070
0x2c	.word 114 "r"	0x00000072
0x30	.word 111 "o"	0x0000006f
0x34	.word 99 "c"	0x00000063
0x38	.word 0x01 ← REL	0x00000001
0x3c	.word use1	0x00000010
0x40	endModule:	

header

one.asm

two.asm

resolved import!

turned old ESR into REL

no more placeholder at 0x10