**The Assembler**

* Assembly code (jr $31) → assembler → machine code (01101…)
* Any translation involves 2 phases
  + **Analysis** – understand what is meant by the source string
  + **Synthesis** – output equivalent target string
* **Assembly file** – stream of characters
  + First group the chars into meaningful tokens
    - E.g. label, hex #, .word, etc.
  + Group tokens into instructions (if possible); aka. Analysis
  + If the tokens do not form sensible instructions, output ‘ERROR’ to stderr
    - Focus on checking correct cases rather than error cases
* How to assemble program branches?
  + Assemble in 2 passes
  + Pass 1 – analysis:
    - Group tokens into instructions
    - Record addresses of labelled instructions in a “symbol table” – list of (label, address) pairs
      * A line of assembly can have more than one label
      * A word can be labelled after the end of a file
    - This pass checks for syntax errors (e.g. improper form/structure) and semantic errors (e.g. defining the same label twice)
    - Output of pass 1: the symbol table & an intermediate representation
      * **Symbol table** – maps labels to addresses
        + E.g. loop → 0x08; end → 0x2c; etc.
        + Can be implemented with a map in C++
      * **Intermediate representation** – remove comments, create tokens, keeps program as ASCII/Unicode chars
    - Pseudocode:

PC = 0

For each line of input

Scan line

For each LABEL

If already in symbol table

ERROR and exit

Else

Add (LABEL, PC) to symbol table

End for

If next token is an OPCODE

If remaining tokens are not as expected

ERROR and exit

Else

Create intermediate representation of instruction

PC += 4

End for

* + Pass 2 – synthesis:
    - Translate each instruction into machine code
    - If an instruction refers to a label, look up the corresponding address in symbol table
    - Output assembled MIPS code to stdout
    - Output symbol table to stderr
    - Pseudocode:

For each OPCODE in the intermediate representation

Translate to MIPS instruction

Look up any labels in symbol table and find address for label

Output instruction (as 4 bytes)

End for

* **Loaders**
* *Missing content…*
* Relocation
  + .word id – add α (load address) to id address (id is a label)
  + .word constant – don’t need to adjust
  + anything else (including beq, bne) – don’t need to adjust
* Assembled file is a stream of bits – how to differentiate between .word (with ids) and instructions?
  + Need more info from the assembler
  + The output of most assemblers is not pure machine code
  + **Object file** – contains binary machine code & auxiliary info needed by the loader & linker
  + Our object file format – MERL – MIPS executable relocatable linkable
  + Need to contain:
    - Machine code
    - Which lines of code (addresses) were originally .word id
* **MERL format** – 3 parts
  + Header – 3 words
    - Always first word: 0x1000 0002 – cookie – indicates this is a MERL file
      * MIPS for beq $0, $0, 2 – i.e. command to skip the header
      * Thus MERL files can be executed as ordinary MIPS files if loaded at 0
    - Second word: length of MERL file (all 3 parts)
    - Third word: code length = length of header + MIPS
  + MIPS binary
    - Assembled to start at α = 0x0c
  + Symbol table (footer)
    - Format code = 1 (indicates relocation)
    - Address (address of a relocatable word in MIPS section)
    - Format code, address, format code, address, …
  + Relocation tool: cs241.merl
    - Input – MERL file & relocation address
    - Output – non-relocatable MIPS file with MERL header & footer removed, ready to load at the given address
  + mips.twoints/array – have optional 2nd argument = address to load the file at
  + E.g. load myobj.merl at 0x1000
    - Java cs241.merl 0x1000 < myobj.merl > myobj.mips
    - Java mips.twoints myobj.mips 0x10000
* Load relocation algorithm

read() //read a word – skip cookie

endMod ← read() – 12 //length of code + footer

codeLen ← read() – 12 //length of code

α ← findFreeRAM(codeLen + stack)

for (int i = 0; i < codeLen; i += 4)

MEM[α + i] ← read()

end for

while (i < endMod)

format ← read()

if (format == 1)

rel ← read() //address to be relocated

//adjust forward by α & backwards by header length

MEM[rel + α – 12] += α – 12

else ERROR

i += 8

end while

* **Linkers**
* Problem: how can the assembler resolve a reference to a label in a different file?
* Solution 1: cat the files together, then assemble the result
* Solution 2: a tool that understands MERL files & can combine them intelligently – a **linker**
* The assembler must be modified to handle references to labels that aren’t there
  + When the assembler encounters .word id where label id is not found, it fills in 0 for id and indicates that the program reuiqres the value of id before it can run