uPOWER Assembler Design

uAssembler for the uPOWER ISA

Outline

- 2 pass Assembler
- Translation of an Instruction
- Actions in Pass 1
- Full assembler

Disclaimer

- uPOWER ISA is in v0.1. There'll be several pieces of information missing
- For this project, take any missing data from the MIPS ISA
  - Eg. Assume that uPOWER ISA memory layout is the same as the MIPS Memory layout

Assembler

- Pass 1
  - Find label definitions. In data and in code.
  - Eg. “varX:”, “loop_begin:”, ...
  - Record position in Code. Relative address is ok.

Assembler – Pass 1

- Pass 1
  - Find label definitions. In data and in code.
  - Eg. “varX:”, “loop_begin:”, ...
  - Record in Symbol Table
    - Record position in Code. Relative address is ok.

Assembler – Pass 2

- Translate every assembly instruction – refer ISA encoding
  - Combine numeric equivalent of opcode, input/output register variables, immediates, and/or labels
- For an instruction using a label:
  - Static Data: calculate offset from Global Pointer. Use in immediate field in the instruction
  - Conditional statement using a label: Eg. beq R3, R4, loop. Read label definition value from SymTab. Calculate offset from current instruction. Use in the instruction

Incremental uAssembler

- One approach is shown
- You can always design what suits you best.

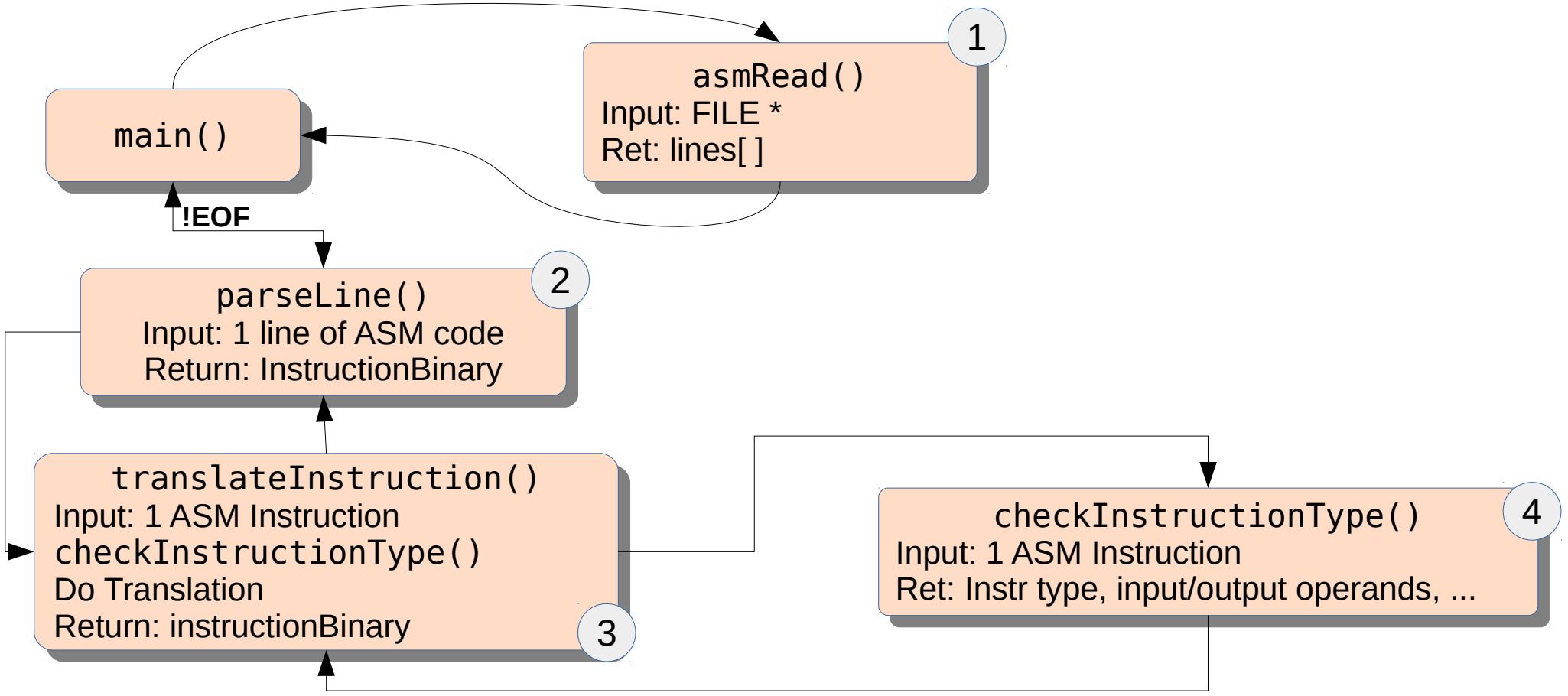
Incremental uAssembler

- Inside out design
- Start from trivial components, build incrementally
  - Step 1 : translate 1 instruction
  - Eg. Instruction: and 4, 5, 6
  - X–type

Incremental uAssembler

v0.1	Identify line type: Instruction/Comment/Assembler Directive/Label
v0.15	Parse a X-type Instruction. Separate out mnemonic, input operands, output operands
v0.2	Translate an X instruction
v0.3	Translate X, XO, instructions
v0.4	Identify .text, .data, and other assembler directives. Do actions correspondingly.
v0.5	First 2 pass code. Identify labels. Count address. Verify.
v0.55	Identify data labels, Build Symbol Table. (in the first pass)
v0.6	Identify data/instruction labels, Build complete symbol table. (in the first pass)
v0.7	Translate D, DS, B, I type instructions (load/store/branches)
v1.0	A fully working uAssembler
>v1.0	Error checking in code, ...

v0.3 Flowchart (Translation)



asmRead()

- Input: uPOWER ISA assembly file
- Output: Strings[ ][ ] (or equivalent)
  - Each line accessible as a string

parseLine()

- Input: A line of uPOWER assembly
  - Check if the line is an instruction/comment/assembler directive/label
  - If instruction and If PASS 2: call translate
  - If comment: ignore
  - If label and If PASS 1: call record addToSymTab
  - If assembler directive and If PASS 1: init data/text pointer, increments, ...
- Output: struct (or equivalent)
  - mnemonic, input operands, output operands
  - Eg. Input: `add 4,5,6`
  - Output: mnemnonic: “add”, in1: “4”, in2: “5”, out: “6”

`parseLine()`
Input: 1 line of ASM code
Return: InstructionBinary

translateInstruction()

- Input: 1 ASM instruction
- checkInstructionType()
- Translate based on the type of instruction
- Output: 32b binaryInstruction

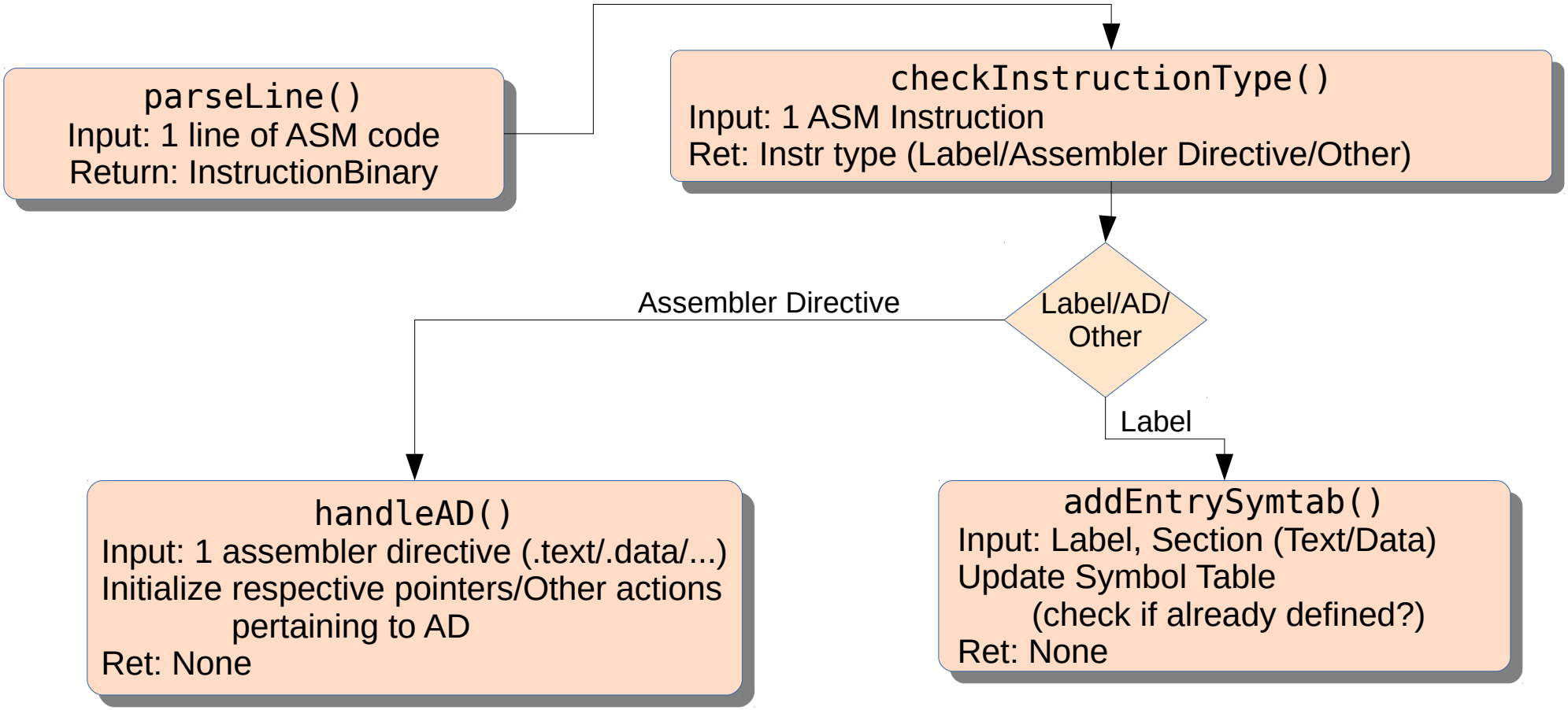
`translateInstruction()`
Input: 1 ASM Instruction
Return: instructionBinary

checkInstructionType()

- Input: 1 ASM instruction
- Output: struct (or equivalent)
  - mnemonic, input operands, output operands
  - Return value of parseLine()

`checkInstructionType()`
Input: 1 ASM Instruction
Ret: Instr type, input/output operands, ...

Pass 1 Flowchart



Translate Loads/Stores/Branches

- B, I, D, DS Instructions
- Get label location from Symtab
- If Branch, Calculate offset from current PC
- If Load/Store: Calculate offset from Global Data Pointer to Variable location
- Any other field to calculate?

uPOWER Functional Simulation

uSimulator for the uPOWER ISA

Functional Simulation

- Reflects the effect of each instruction on the System, faithfully
- Accurate representation of dynamic instruction count, memory accesses, INT/FP ALU operations, branches, function calls
- Does not keep track of timing information

Structures to track in uSim

- Register file, Special registers (?), Memory
- Before program begins execution:
  - Initialize memory
  - Initialize register file

Structures to track in uSim

- During program execution:
  - Load instructions read from Memory
  - Store instructions modify Memory
  - ALU instructions read and modify RF
  - Control instructions modify special registers (NIA, CIA, ...)

uSim Functional Simulation

- Initialize Memory, RF, special registers, CIA, NIA, ...
- Put the output of the Assembler (binary code) in the Memory
- Read one instruction from the memory from NIA
- Interpret (Decode) the instruction based on Opcode and other fields
- Extract input and output operands
- Read input operands from the respective structures
- Execute the instruction
- Write the output in the relevant structures
- Change PC (NIA) – Caution if current instruction is a Control statement

Final Thoughts

- High level approach is presented
- You are free not follow any of this
- More detailed implementation will get bonus

Demo 1

- Demo of the working of the Simulator can be done twice
- Mandatory Demo 1 – Feb 17. Evaluation will be done for this demo only.
  - Write uPOWER equivalents of the 4 programs from the SPIM tutorial
  - Show code assembly (for at least these 4 uPOWER programs)
  - Show code simulation (stepwise code execution demo)

Demo 2

- Optional.
- If you pursued simulator development beyond what is required for this assignment, I’d be glad to see how it turned out.
- Show at the end of the Semester.