**Computer Structure – Assembler / Simulator Documentation**

**ASSEMBLER FLOW**

Program.asm

Imemin.txt

Dmemin.txt

**Func explanations**

**void init\_DS();**

allocate memory for local use arrays and queues. The arrays will be copied to output files and queues are used for updating instructions with labels.

**int fix\_line\_for\_parsing(char\*\* broken\_res, char \*lineBuffer);**

gets line as it was read from asm file as lineBuffer, break it down to an array of strings as broken\_res, without comment if there is any.

**void parseLine(char \*\*line, int lLength, int \*pc, int \*dtable\_size);**

gets line and its length from fix\_line\_for\_parsing as line and lLength, determine its type, sending it to corresponding parsing function, and updates pc or dmemin local array biggest index if necessary.

**void update\_labled\_instructions();**

after parsing all lines from asm file and writing it into local arrays, scan queue which is (1) instruction’s pc which has label and the desired label name and update the label by scanning other queue which is (2) label’s name and its corresponding pc.

**int break\_buffer(char \*\*broken\_buffer, char \*lineBuffer);**

gets line as it was read from asm file as lineBuffer and break it down to an array of strings according to a set of delimiters into broken\_buffer. Returns the amount of strings contained in broken\_buffer

**void drop\_comment(char \*\*line, int \*line\_len);**

search line, which is the result of break\_buffer, for ‘#’, drop all strings from this point onward and update the amount of strings in line.

**lineType get\_lineType(int line\_len);**

determine which kind of line was passed into parseLine according to the line length. After fix\_line\_for\_parsing there is one-to-one match between line’s length and its type.

**void parseInstruction(char\*\* line, int pc);**

gets line and pc, generating the parsed version of line into temporary string, a copying it to local array at pc index.

**void parseLabel(char \*label\_name, int pc);**

gets label’s name and corresponding pc, making a new queue’s node and adding it to label’s queue for later addressing.

**void parseWord (char \*\*line, int \*dtable\_size);**

gets pseudo-command to add word as line, adding data to local dmem array at address index.

**void update\_pc(int \*pc, lineType lt);**

gets current pc and kind of instruction that was just parsed, updating pc if needed according to lineType.

**void update\_dtable\_last\_idx (int \*dtable\_size, int curr\_address);**

gets current biggest index known in dmem local array and current index from read word-line and updating the biggest index if needed.

**void add\_op\_to\_result(char\* result, char\* str);**

gets str which is the op code as it was read from asm file, converting it to parsed version and copying it into result. Copy and not concatenating because it is the first string inside parsed version of line.

**void add\_reg\_to\_result(char\* result, char\* str);**

getting a register as it was read from asm file as str and concatenating its parsed version into result.

**int add\_imm\_to\_result(char\* result, char\* str);**

getting an immediate as it was read from asm file as str and concatenating its parsed version into result. If it’s a label, concatenate a flag for later use, and make a node of this instruction for instructions\_with\_label queue for later addressing.

**void str2param (char\* result, const char \*str);**

gets str which is op or register as it read from asm file and convert it to its parsed version, concatenate it into result. Result size vary due to sizes’ indifference of register and opcode.

**int isLabel(char \*str);**

gets a str which is immediate as it was read from asm file and determine if it’s a label by checking its first character.

**int isHexa (char\* str);**

gets a string representation of a number as str and checks if it’s a hexadecimal representation by checking if its prefix is 0x / 0X.

**int str\_to\_2complement(char\* str, int hex\_len);**

gets a decimal number (positive or negative) as str and returns the number which is the equivalent of str in 2’s complement for an integer with hex\_len\*4 bits.

**void num2hexa (char \*result, char \*str, int hex\_len);**

gets a number as str and returns its hexadecimal representation into result with hex\_len bytes.

**void make\_new\_node(data\_node \*new, char \*name, int pc);**

gets a pointer to a new node and fills it with data according to inputs.

**void add\_to\_Queue(data\_node \*new, int isLabel);**

add the node new to queue. Queue is determined by flag that is passed by input

**data\_node\* find\_node(char\* search\_term, int LabelQ);**

returns pointer the node that contains search\_term in the queue which is determined by the flag from the input.

**void free\_Queues();**

free the memory allocated in both queues. The function called after the queues’ work is done.

**void add\_line\_to\_table(char \*parsedLine, int hex\_address, int is\_intruction);**

gets parsed line to be copied into local array in address index. The table is determined (dmem/imem) by the flag from input.

**void copy\_table\_to\_file(char \*\*table, int table\_size, char \*file\_path);**

gets local array and its size and copies it into the file which is named file\_path.

**void free\_table(char \*\*table, int start, int end);**

free the local array pointed by table in range [start, end).

**int init\_unparsed\_instructions(FILE \*fp, int len);**

gets file pointer to program.asm file and len for each line’s buffer and copy its content into local array. Returns the amount of lines read from the asm file.

**SIMULATOR FLOW**

Dmemout.txt

Diskout.txt

Regout.txt

Trace.txt

Hwregtrace.txt

Cycles.txt

Leds.txt

Display7seg.txt

Monitor.txt

Monitor.yuv

Imemin.txt

Dmemin.txt

Irq2in.txt

Diskin.txt

**Func explanations**

**void add\_to\_cmd\_lst(Instruction \*cmdLst, char \*inst);**

This function gets hexadecimal instruction from imemin.txt, and parses it to struct instruction, with decimal fields for opcode, registers, and immediates.

**void main\_loop();**

Called after initiating all vars and structures – and it runs on cmdLst, calls handlers, and calls run\_command for execution – main\_loop is the second part of the main() function. Main() inits variable, and main\_loop runs main flow.

**void interrupt\_handler();**

Checks if any interrupts have been called, and if so, runs the flow needed to handle them (Whether it is changing PC, calling IO devices, etc.)

**void update\_irqs\_state(int \*irqState);**

Updates irq IO Registers – checks if an interrupt has been given.

**void diskIO\_handler();**

This function handles reads/write from/to disk – checks disk cycles, and writes/reads after 1024 cycles.

**void timer\_handler();**

Handles timer – checks if timer needs to be increased, and that it hasn’t reached the limit.

**void update\_monitor\_pixels();**

Updates current monitor pixels, based on information for monitor IO registers.

**void update\_irq2(int cycle);**

Checks if irq2 reached the next cycle it calls an interrupt, and handles it.

**int run\_command(Instruction instruction);**

Executing given command – separates R/I format operations, and calls needed func to execute current opcode.

**void run\_arithmetic(Instruction instruction, int id);**

This function is called by run\_command if the next command is an arithmetic command – executes it on needed registers.

**void run\_jump\_branch\_commands(Instruction instruction, int id);**

This function is called by run\_command if the next command is a branch command – executes it on needed registers.

**void run\_memory\_command(Instruction instruction , int id);**

This function is called by run\_command if the next command is a memory command – executes it on needed registers.

**void run\_IOregister\_operation(Instruction instruction , int id);**

This function is called by run\_command if the next command is an in/out command – executes it on needed IO registers.

**void init\_txt\_files();**

Creates required txt files, so they can be appended upon change (for files that change along the run, such as leds, display7seg, etc.).

**void write\_exit\_txt\_files();**

Called when program terminates successfully – spills all given information to .txt files as required in the assignment.

**Simulator / Assembler Util functions**

**/\*-----------Read Functions-----------\*/**

**int read\_from\_file(FILE \*fp, int len, Mode mode);**

**int init\_disk\_lst(FILE \*fp, char \*line, int len);**

**int init\_data\_lst(FILE \*fp, char \*line, int len);**

**int init\_inst\_lst(FILE \*fp, char \*line, int len);**

**int init\_irq2\_lst(FILE \*fp, char \*line, int len);**

**int add\_to\_inst\_lst(char \*instruct, char \*line);**

**int add\_to\_data\_lst(int \*mem, char \*data);**

**int add\_to\_irq2\_lst(int \*irq2, char \*data);**

**void read\_from\_disk(int\* disk\_sector, int\* mem\_buffer, int buffer);**

**/\*----------Write Functions-----------\*/**

**int write\_to\_file(FILE \*fp, int len, Mode mode);**

**int write\_int\_arr\_to\_file(FILE \*fp, char \*line, int line\_len, int \*arr, int arr\_len);**

**int write\_str\_to\_file(FILE \*fp, char \*line);**

**int write\_diskout(FILE \*fp, char \*line, int len);**

**int write\_dmemout(FILE \*fp, char \*line, int len);**

**int write\_registers(FILE \*fp, char \*line, int len);**

**int write\_trace(FILE \*fp, char \*line, int len);**

**int write\_hwregtrace(FILE \*fp, char \*line, int len);**

**int write\_led\_7seg(FILE \*fp, char \*line, int len, int IORegIndex);**

**int write\_cycle(FILE \*fp, char \*line, int len);**

**int write\_monitor(FILE \*fp, char \*line, int len);**

**void write\_to\_disk(int\* disk\_sector, int\* mem\_buffer, int sector);**

These function’s purpose is reading/writing to txt files – reading when simulator starts running to have imemin/dmemin/diskin/irq2in as variables so the program can run its main loop more fluently and writing when program terminates to output .txt files.

**void fill\_with\_null(int start, int end, Mode mode);**

Fills given (mode) variable/struct with nulls, from [start] to [end].

**char cut\_string\_by\_index(char \*str, int i);**

Cuts given str on index i.

**int compare (const void \* a, const void \* b);**

compares obj a and b, for qsort func.

**void sign\_ext(int \*num);**

Performs sign extension to num.

**void dec2hexa(char\* result, int num, int len);**

Converts num from dec to hexa, and writes output in result.

**int hexa2dec(char \*hex\_rep, int len);**

Converts hex\_rep from hexa to dec, and returns result as an integer.

**void set\_line\_to\_zero(char \*line, int len);**

Sets line from index 0 to len to zeros.

**int get\_max(int a, int b);**

returns max between a and b.