תיעוד פרויקט בקורס ארכיטקטורה של מחשבים

204620702 205431224 208410969

קובץ core.h - הקובץ שמגדיר ליבה(ממומש ע"י **core.c**)
הקובץ מגדיר את כל הפונקציות הממומשות בcore.c. הקובץ מגדיר את הפונקציות שאחראיות לסמלץ מחזור שעון, לסמלץ כל שלב בpipeline, לסמלץ מעבר של המידע בין שלבי הpipeline בסוף כל מחזור שעון ולבצע את כל ההדפסות הדרושות.
לצורך המימוש הוגדרו הMACROS הבאים:

- כל הopcodes הנתמכים ע"י המעבד:

```
#define ADD  0
#define SUB  1
#define AND  2
#define OR   3
#define XOR  4
#define MUL  5
#define SLL  6
#define SRA  7
#define SRL  8
#define BEQ  9
#define BNE  10
#define BLT  11
#define BJT  12
#define BLE  13
#define BGE  14
#define JAL  15
#define LW   16
#define SW   17
#define HALT 20
```

- כל הפרמטרים הנדרשים על מנת לבצע decoding נכון של הפקודה:

```
#define OPP_SHFT 0x18 //24 bits
#define RD_SHFT 0x14 //20 bits
#define RS_SHFT 0x10 //16 bits
#define RT_SHFT 0xc //12 bits
#define OPP_MASK 0xFF000000 //bits 31:24
#define RD_MASK 0x00F00000 //bits 20-23
#define RS_MASK 0x000F0000 //bits 16-19
#define RT_MASK 0x0000F000 //bits 15-12
#define IMM_MASK 0x00000FFF //bits 11-0

#define SRL_MASK 0x7FFFFFFF
#define LSB_9BIT 0x000001FF
#define MEM_SIZE 1024
```

כמו כן הוגדרו מבני העזר הבאים:

```
typedef struct core {
    FILE* trace_file;
    int is_halt;
    int pc;
    int next_pc;
    int data_hazzard;
    int prev_cache_miss;
    int prev_mem_inst_pc;
    int Op_Mem[MEM_SIZE];       //Operation Memory aka IMEM
    int Reg_File[16];

    operation *IF_op;           //Operation in Fetch
    operation *ID_op;           //Operation in Decode
    operation *EX_op;           //Operation in Execute
    operation *MEM_op;          //Operation in Memory
    operation *WB_op;           //Operation in Write Back

    cache *Cache;               //Cache

    core_stats *core_stats;     // single core statisrics

} single_core;
```

```
single_core* cores[cores_count]; // array of cores in the CPU
```

```
typedef struct operation {
    int code; //the code of the operation
    int rd; //rd register
    int rs; //rs register
    int rt; //rt register

    int rd_val; // value of rd @Decode
    int rs_val; // value of rs @Decode
    int rt_val; // value of rt @Decode

    int addr; // used in memory instructions

    int empty; //indicates a bubble;

    int op_pc;
    int inst; // the opcode line from Imem
    void(*op_code)(int core_num); // a function pointer according to opcode
    } operation;
```

```
typedef struct stats{
    int cycles;
    int instructions;
    int read_hit;
    int read_miss;
    int write_hit;
    int write_miss;
    int decode_stall;
    int mem_stall;
} core_stats;
```

הפונקציות המוגדרות בקובץ זה:

- פונקציות של איתחול או כתיבה לקבצי פלט:

```
/* ************************************************************/
/*  ~~~~~~~~~~~  Initializing and print functions ~~~~~~~~~~  */
/* ************************************************************/
void init_operation(operation* op);

void init_core(FILE* trace_file, FILE* imem, single_core* core, int core_num);

void read_imem(FILE* imem, single_core* core);

void print_trace(single_core* core, int clock_cycle);

void print_regs(single_core* core, FILE* regs);

void print_stats(int core_num, single_core* core, FILE* regs);
```

- פונקציות המתארות את הפונקציונליות של כל opcode

```
/* ********************************************************/
/*  ~~~~~~~~~~~~~~    SIMP OP CODES OPERATIONS ~~~~~~~~~~~~~~  */
/* ********************************************************/
//1
void add(int core_num);

//2
void sub(int core_num);

//3
void and(int core_num);

//4
void or (int core_num);

//5
void xor (int core_num);

//6
void mul(int core_nume);

//6
void sll(int core_num);

//7
void sra(int core_num);

//8
void srl(int core_num);

//16
void lw(int core_num);

//17
void sw(int core_num);

//assign to $0 or unreconized behivour
void nop(int core_num);
```

- פונצקיות המתארות את הפונקציונליות של הליבה עצמה וכל שלב בpipeline

```c
/* ************************************************************/
/*  ~~~~~~~~~~   Single core execution functions ~~~~~~~~~~  */
/* ************************************************************/

void IF_ex(single_core* core);

//void fetch_op(single_core* core);

int ID_ex(single_core* core);

void EX_ex(int core_num);

int MEM_ex(single_core* core);

int WB_ex(single_core* core);

/* Simulates Single Core clock cycle.
receives the core, the number of the click cycle and a pointer to int that indicates wheter to end the execution of this core*/
void simulate_clock_cycle(int core_num, int clock_cycle, int* halt);

void end_clock_sycle(single_core* core, int data_hazzard);

int detect_data_hazzard(single_core* core);
```

קובץ memory.h - הקובץ שמגדיר את מערכת הזיכרון (ממומש ע"י **memory.c**)

קובץ המכיל את הגדרות הפונקציות הרלוונטיות לניהול מערכת הזיכרון (ובפרט את הממשק שלה עם הליבות). הנוסף מכיל הקובץ הגדרות מאקרו, אשר מיועדות לשמש Enumerators או קבועים עבור משתנים שונים. בנוסף בקובץ מוגדרות מספר פונקציות מאקרו, בעיקר עבור מיפוי כתובות, או פעולות קטנות לצורך כך שהקוד יהיה בהיר. כן יצורפו החלקים המהותיים להבנת אופן ההתנהלות.
מבחינה כללית: בממשק בין הליבה למטמון (וממנו לזיכרון) הוגדר באופן הבא: הליבה מבקשת מהזיכרון לבצע פעולות זיכרון על כתובת מסוימת ע"י פונקציות מתאימות (write, read). במידה והמידע אינו שמור בstate המתאים (MISS) מתבצע stall בליבה.
במצב זה הליבה תבקש שוב מהמטמון את אותה הבקשה שוב ושוב, אך הוא ידע להבחין שהוא כבר מטפל בה, ולכן למעשה לא יבצע דבר. כאשר המידע יגיע, ויאוחסן במטמון הוא יחזיר לליבה HIT והיא מבחינתה, תוכל להתקדם.

פקודות המאקרו שאחראיות למיפוי הכתובת למרכיביה השונים:

```
#define _get_tag(address)    ((((unsigned int)address) & TAG_MASK) >> TAG_SHFT)
```

```
#define _get_block(address) ((((unsigned int)address) & BLK_MASK) >> BLK_SHFT)
```

```
#define _get_idx(address)    (((unsigned int)address) & IDX_MASK)
```

```
#define alligned(address)    (((unsigned int)address) - (((unsigned int)address) % BLK_SIZE))
```

```
#define construct_address(tag,idx)    (alligned(((tag << TAG_SHFT) | (idx))))
```

פקודות המאקרו עבור ביצוע פעולות פשוטות, המסייעות לקוד להיות בהיר יותר:

```
#define _cache_handled(handler) (handler < 4)
```

```
#define inc_positive(time) ((time >= 0) ? (time + 1) : time)
```

```
#define _cache_on_bus(idx,clock_cycle) (last_time_served[idx] = clock_cycle)
```

```
#define time_diff(time_a, time_b) (time_a - time_b)
```

```
#define is_hit(st) (st == HIT)
```

```
#define is_miss(st) (st == MISS)
```

```
#define evict_first(c) ( (pending_evc[c] != NULL) & (pending_req[c] != NULL) )
```

```
#define need_to_evict(blk, c) (c->mesi_state[blk] == Modified)
```

לצורך מימוש מערכת הזיכרון נכתבו מבני הזיכרון הזיכרון המתארים מטמון, זיכרון ראשי וה MESI Bus. בנוסף נכתבו מבני עזר שמסייעים לביצוע הסימולציה.

```
typedef struct cache{
    int cache_data[WORDS];    // the actual data stored by cachelines (word)
    int tags[BLOCKS];         // the tags of the stored blocks
    int mesi_state[BLOCKS];   // the mesi status of the stored blocks
    int idx;                  // serial id of cache instance
    int busy; // a flag used by cache to indicate that the cache request is being processed.
    bus_request_p next_req;      // store the request that will be send on MESI BUS
    bus_request_p next_evict;    // store evict if such is needed
} cache ;
typedef cache* cache_p; // pointer to cache
```

```
typedef struct bus_request{
    int cmd;                    // command type
    int addr;                   // requested address
    int data;                   // data
    int id;
} bus_request;
typedef bus_request* bus_request_p; // pointer to bus_request
```

```
typedef struct response{
    int handler;                // handler of current request
    int requestor;              // origin of current request
    int copyed;                 // counter of transaction in current request
    int cmd;                    // current request type (to remember when flushing)
} response;
typedef response* response_p; // pointer to response
```

```
typedef struct mesi_bus{
    int state;                  // for implementation
    int origin;                 // the origin of transaction
    int cmd;                    // type of transaction
    int addr;                   // address
    int data;                   // data
    int shared;                 // Shared state indicator
    int req_id;                 // request_id (for debugging)
    response_p resp;            // used for flush
} mesi_bus;
typedef mesi_bus* mesi_bus_p; // pointer to mesi_bus
```

```
// A struct for implementation of main memory
typedef struct main_memory{
    int data[mem_size];
    int latency;
} main_memory;
typedef main_memory* main_memory_p; // pointer to main_memory
```

בנוסף, הוגדרו משתנים גלובליים אליהם יש גישה מכל פונקציה: מערך המטמונים (אינדקס בהתאם לליבה אליה הוא שייך), הזיכרון הראשי וה MESI bus. בנוסף מערך של בקשות ממתינות, פינויים ממתינים, ומערך last_time_served המשמש לצורך מימוש Round Robin וכן משתנה המודד את הזמן אותו אנחנו ממתינים כבר לזיכרון הראשי.

```
cache_p CACHES[CACHE_COUNT];       // array of caches
main_memory_p Memory;              // main memory
mesi_bus_p Bus;                    // the main mesi bus
bus_request_p pending_req[CACHE_COUNT]; //all the pending requests
bus_request_p pending_evc[CACHE_COUNT]; // all the pending evicts
int last_time_served[CACHE_COUNT];      // array for use of round robin
int waited_cycles;                      // counter when accessing main memory;
```

הפונקציות המופיעות בקובץ מחולקים למקטעים לפי שימושים שלהם:
המקטע הראשון הוא עבור פונקציות של הקצאת זיכרון או כתיבת קבצים:

```
// Allocate cache fields
void init_cache(cache_p cache);
```

```
// Allocate memory strict & MESI bus
void initiate_memory_system();
```

```
// free memory struct & MESI bus
void close_memory_system();
```

```
// free cache fields & pointer
void release_cache(cache_p cache);
```

```
// print bus trace file
void write_bus_trace(FILE* file_w, int currect_cycle);
```

```
// print memory in 'dump' format
void dump_cache(cache_p c, FILE* dsram, FILE* tsram);
```

```
// print all memory components memory in 'dumb' format
void dump_memory(sim_files_p files_p);
```

המקטע השני הוא פונקציות הנוגעות לממשק בין הליבה לזיכרון. כל ליבה, מבצעת קריאה / כתיבה למטמון שלה.

```
// check if address is cached, return HIT or MISS
// for BusRd - if data is valid it's enough
// for BusRdX - data has to be in M state
int query(int address, cache_p cache, int mode);
```

```
// read word from cache. if MISS, fetched it through messi and stall
int read_word(int address, cache_p cache, int* dest_reg);
```

```
// write data to cache. if MISS, fetched it through messi and stall
int write_word(int address, cache_p cache, int* src_reg, int pc);
```

המקטע השלישי הוא פונקציות הנוגעות לניהול פרוטוקול הMESI, והתקשורת בינו הBus למטמונים:

```
// call upon loading request on the bus
void clear_request_from_cahce(int c, int clock_cycle);
```

```
// load a transaction on the bus (request)
void generate_transaction(bus_request_p request, int clock_cycle);
```

```
// check if data is also cached in other caches
```

```
int is_shared(int requestor, int address);
```

```
// determine the handler of the request: if (stored & modified) handler = cache.
void set_handler(int address);
```

```
// call when a request is loaded on the bus
void kick_mesi();
```

```
// no one uses the bus if waiting for memory
void wait_for_response();
```

```
// transfer request over the line
void flushing(int clock_cycle);
```

```
// go over caches, and invalidate the data if needed, skip given caches
void invalidate_caches(int client, int provider, int block_idx);
```

```
// perform memory copy on flush when request was Rd
void snoop_Rd(int handler, int client);
```

```
// perform memory copy on flush when request was RdX
void snoop_RdX(int handler, int client);
```

```
// perform memory copy on evict
void evict();
```

```
//snoop the line on flush: modify memory elements
void snoop();
```

```
// get the next core to serve
int next_core_to_serve(int clock_cycle);
```

```
// chose next command on bus (used on dirty evict)
bus_request_p get_request_per_cache(int cache_idx);
```

```
// when evict is no longer needed
void clear_old_evicts();
```

```
// get longest request waiting
bus_request_p get_next_request(int clock_cycle);
```

```
// manage transaction over messi using state machine
void mesi_state_machine(sim_files_p files, int clock_cycle);
```

המקטע הרביעי (אינו מצורף) מכיל פונקציות ששימשו אותנו לצור debugging

קובץ sim.h

קובץ המכיל ההגדרות לפונקציות שאחראיות על הריצה הכללית של הסימולטור. בקובץ זה ניתן למצוא הגדרה של מצביע גלובלי למבנה הקבצים של הסימולטור. כמו כן ניתן למצוא בו הצהרות של הפונקציות הבאות :

```
void init_cores(single_core** cores);
```

Function to initiate all 4 cores structures.

```
void init_main_memory(FILE *memin);
```

Function to initiate the simulator’s main memory using the memin.txt file.

```
void init_cores_done(int **cores_done);
```

Function to initiate an array to identify if a core is done running (halt op was executed). In the beginning the array contain only zeros, when a core is done in a cycle the matching array index will turn to 1.

```
int main(int argc, char* argv[]);
```

Main simulator function.

קובץ files.h

קובץ המכיל את הגדרת מבנה ה-FILES בו משתמשות פונקציות שונות מהפרויקט, לכן החלטנו לעשות אותו כמבנה גלובלי שניתן לגשת אליו מכל מקום בתוכנית.

```
typedef struct sim_files{
    FILE *imem0;
    FILE *imem1;
    FILE *imem2;
    FILE *imem3;
    FILE *memin;
    FILE *memout;
    FILE *regout0;
    FILE *regout1;
    FILE *regout2;
    FILE *regout3;
    FILE *core0trace;
    FILE *core1trace;
    FILE *core2trace;
    FILE *core3trace;
    FILE *bustrace;
    FILE *dsram0;
    FILE *dsram1;
    FILE *dsram2;
    FILE *dsram3;
    FILE *tsram0;
    FILE *tsram1;
    FILE *tsram2;
    FILE *tsram3;
    FILE *stats0;
    FILE *stats1;
    FILE *stats2;
    FILE *stats3;
}sim_files;
typedef sim_files* sim_files_p;
```

```
sim_files_p init_files_def(sim_files_p files);
```

Function to initiate the sim_files structure when there wasn't any files names given, as demanded on the project's instructions.

```
sim_files_p init_files(sim_files_p files, char* argv[]);
```

Function to initiate the sim_files structure when all files names are given as the program's input.