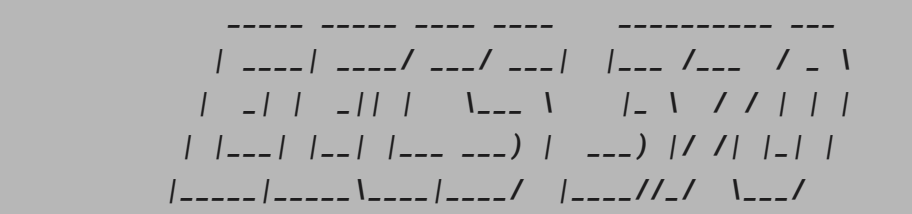
Final Exam



EECS 370 Fall 2021: Intro to Computer Organization

Regarding the exam:

- You are allotted one **8.5 x 11 double sided** note sheet as well as any additional reference material provided within the exam.
- Calculators without wireless are allowed, but ***no cell phones or internet connected devices.***
- **Any work written on a blank page will be ignored.** ***Any work on a blank side of a sheet is not guaranteed to be scanned or graded.***
- You have **120 minutes** to complete the exam. Don't spend too much time on one question.
- There are **9** questions in the exam on **25** pages. Point values for each problem are posted at the beginning of each section. There are **3** pages of reference material. **Please flip through your exam and ensure that you have all 25 pages.**
- Partial credit will be awarded based on work shown.
- Write ***legibly and dark enough*** for the scanners to read your work.
- ***Please write your unigname on the line provided at the top of each page***

You are to abide by the University of Michigan College of Engineering honor code. Please sign below to signify that you have kept the honor code pledge:

I have neither given nor received aid on this exam, nor have I concealed any violations of the Honor Code.

Question 1: Warmup *{ 10 Points }*

Indicate whether the following statements are true or false ***(1pt each)***

True	False	Statement
○	○	If a given benchmark achieves a higher CPI on datapath A than datapath B, then the benchmark will always take longer to execute on datapath A than datapath B.
○	○	When handling data hazards in a pipelined datapath, detect-and-forward always leads to a CPI less than or equal to that of detect-and-stall.
○	○	Consider a cache with four 1-byte blocks for a byte-addressable machine. Assume the cache is initially empty. It is possible for a direct-mapped version of the cache to have a hit while a fully-associative version of the cache would have a miss for the same access.
○	○	Having multi-level page tables vs a single-level page table gives modern processors the ability to access physical memory faster.
○	○	Caller save always leads to a lower number of register load/store pairs being executed than callee save.
○	○	The symbol table tracks which instructions need to be updated after linking.

Circle the correct answer choice: ***(1pt each)***

1. If a data cache does not contain a dirty bit, then it must be using a _____ policy.

   write-through ***write-back***

2. _____ is the concept that the likelihood of referencing a piece of memory is higher if an access has occurred near it.

   spatial locality ***temporal locality***

3. **Short Answer:** Is LC2K Big or Little Endian? Please write your answer in 15 words or less. **(2pts)**

Question 2: Pipeline Cache Benchmark ***{ 12 Points }***

Consider the pipeline from lecture with times below, using detect and forward for data hazards and speculate and squash for control hazards.

- Data Memory Read/Write: 30 ns
- Instruction Memory Read: 30 ns
- ALU: 50 ns
- Register file read/write: 5 ns
- All other operations, wire transfer: 0 ns

Say you have an LC2K program with ***1000*** instructions and has the following characteristics:

- 30% of instructions are lws
- 10% are sws
- 15% are adds
- 20% are nors
- 25% are beqs
- 20% of the instructions are immediately followed by an instruction dependent on it.
- 10% of the instructions are followed by an independent instruction and then by a dependent instruction.
- There are no other dependencies.
- You may assume that the above about data hazards is true no matter what instructions are involved.
- 70% of branches are not taken

1) Assume that we use predict-not-taken for branches, what is the runtime for the program? ***(3.5pts)***

Answer: _____ ns

2) Say we have another design that separates our execution stage and memory stage into two stages each ***(IF, ID, EX1, EX2, MEM1, MEM2,*** *and* ***WB).***

- The 50 ns **ALU** operation is divided into two 25 ns operations.
- Data is always forwarded to **EX1**.
- **lw** and **sw** instructions will finish in **MEM2**.
- **add** and **nor** instructions will finish in **EX2**
- Branches always resolve in **MEM1**.

What is the runtime for the program now? ***(3.5pts)***

Answer: ns

3) Let's assume that we know that **80 % of cache accesses** are hits and **99.99 %** of main memory accesses are hits. If cache latency is **5 ns**, main memory access latency is **200 ns**, and disk latency is **10,000 ns**, what is the average access time to memory? Assume that everything we want to access would be in the disk. ***(2.5pts)***

Answer: ns

4) Assume that it remains true that **99.99 % of main memory accesses are hits**, and all the latencies stay the same as the previous question. What is the threshold for cache hit rate that implementing a cache would help reduce memory access time? Choose > or <, fill out the blank and show your calculations. ***(2.5pts)***

Hit rate (> / <) _____***%***

Question 3: Branch Predictions ***{ 10 Points }***

```
...
loop add   1    2    1
...
     beq   0    5    skip
...
skip noop
     beq   0    1    done
     beq   0    0    loop
done halt
```

You are given the following LC2K code, with some assembly omitted. Answer the following questions about branch prediction with the information provided.

You are to assume when predicting taken, the processor knows which address the branch branches to.

a) We have a **static predictor** which predicts **backwards taken and forwards not taken** for each branch locally. You are provided with the local history of each branch. Fill in the following table:

Instruction	***Local History***	***# Of Mispredicts*** (1 point each)
beq 0 5 skip	***T, N, T, N, T***	
beq 0 1 done	***N, N, N, N, T***	
beq 0 0 loop	***T, T, T, T***	

b) We have a **2 bit saturating counter** for **local branch prediction** which is initialized at **strongly not taken**. You are provided with the local history of each branch. Fill in the following table:

Instruction	***Local History***	***# Of Mispredicts*** (1 point each)
beq 0 5 skip	***T, N, T, N, T***	
beq 0 1 done	***N, N, N, N, T***	
beq 0 0 loop	***T, T, T, T***	

c) We have a **1 bit counter** for **global branch prediction** which is initialized to **not taken**. How many mispredicts are there? The global history has been provided for your convenience. (2 points)

Global History	***T***	***N***	***T***	***N***	***N***	***T***	***T***	***N***	***T***	***N***	***N***	***T***	***T***	***T***

 # Of Mispredicts:

d) Which predictor gives us the best performance? Assume the cycle time and mispredict penalty are the same for all 3 predictors. Circle the predictor with the best performance.

Static Predictor ***2 Bit Local Predictor*** ***1 Bit Global Predictor***

Question 4: 3 C's of Cache ***{ 9 Points }***

James, a student in EECS 370, is trying to classify the different types of cache misses by running a program that traverses through an array. **The *CACHE_ACCESS* function returns *CACHE_MISS* or *CACHE_HIT* depending on if the access is a miss or hit in the cache.** ***CACHE_ACCESS reads in an integer starting at address j. DATA array starts at address 0.***

```
int DATA[64]

float traverse_array(int stride){
     int misses = 0;
     for(int i = 0; i < stride; i += 4){
          for(int j = i; j < 256; j += stride){
               if(CACHE_ACCESS(j) == CACHE_MISS)){
                    ++misses;
               }
          }
     }
     return misses / 64.0;
}
```

The 370 cache specification is listed below:

- ***Byte-addressable***
- ***Cache size: 128 B***
- ***Block size: 8 B***
- ***4-way set associative***
- ***Write-back, allocate-on-write***
- ***LRU replacement policy***

a) What would be the miss rate for the listed values of **stride** in an **infinitely-sized cache**? (3pts)

Stride	***Miss rate*** (each blank worth 1 pt)
4	
8	
16	

b) What would be the miss rate for the listed values of **stride** in a **fully-associative cache (of same size as the 370 cache)**? (3pts)

Stride	***Miss rate*** (each blank worth 1 pt)
4	
8	
16	

c) What **proportion of ALL misses** would be **conflict misses** in the **370 cache**?
(Hint: use your calculation/work from the infinitely-sized cache and think about the behavior of the FA cache) ***(3pts)***

Stride	***Conflict miss rate given any type of miss (i.e. conflict misses/all misses)*** (each blank worth 1 pt)
4	
8	
16	

Question 5: Data Hazards ***{ 14 Points }***

Consider the program below:

Address	*Label*	*Opcode*	*Arg0*	*Arg1*	*Arg2*
0		lw	0	1	one
1		lw	0	3	one
2	eecs	add	3	1	3
3		lw	0	2	two
4		beq	3	2	eecs
5		add	1	1	4
6		lw	0	5	three
7		beq	2	3	2
8		nor	1	1	6
9		add	1	6	1
10		add	5	1	5
11		lw	0	4	one
12		add	2	1	4
13		beq	4	3	skip
14		lw	0	5	one
15	skip	add	4	2	1
16		nor	1	5	3
17		halt			
18	one	.fill	1		
19	two	.fill	2		
20	three	.fill	3		

Recall the pipeline from lecture:

IF → ID → EX → MEM → WB

Suppose you use a modified version of the lecture pipeline. To improve cycle time, you create a new pipeline that has two ***MEM stages, MEM1 and MEM2, and two EX stages, EX1 and EX2.*** Now, the pipeline is:

IF → ID → EX1 → EX2 → MEM1→ MEM2 → WB.

- Data is always forwarded to ***EX1***.
- Data can only be forwarded from ***MEM1*** and ***WB***.
- **lw** and **sw** instructions will finish in ***MEM2***.
- **add** and **nor** instructions will finish in ***EX2***.
- Branches always resolve in ***MEM1***.
- Register file has internal-forwarding functionality

a) Select all registers that cause a RAW (Read After Write) data hazard. Assume data hazards and control hazards are both resolved using **detect and stall**. Remember, *data hazards can only occur from instructions that are executed.* ***You may not need all the lines provided below.***

(4 pts)

Register	***Source Line***	***Dependent Line***

b) Using **detect and stall** to resolve data hazards, and **detect and stall** to resolve control hazards, what is the number of stalls that occur due to **data hazards**? **(3pts)**

Answer:

c) Using **detect and forward** to resolve data hazards, and **predict taken for forward branches** and **predict not taken for backwards branches** to resolve control hazards. Answer the statements below.

i. How many cycles are required to empty out the pipeline (at the end of a program)? ***(1 pts)***

Answer:

ii. How many instructions are run in this program? ***(2 pts)***

Answer:

iii. With parts (i) and (ii) in mind, how many cycles does the program take to run? ***(4 pts)***

Answer:

Question 6: Reverse Engineering a Cache ***{ 10 Points }***

You have just purchased a new Macbook Pro and having just taken 370, you know that the cache plays an important role in the speed of the processor. You decide you want to find the exact specifications of the cache.

- You may assume a ***64-bit byte addressable system.***
- You may assume that the cache has the same structure as the caches we have discussed in 370.
- You may assume that the block size, blocks per set, and number of sets are all powers of 2.
- You may assume that the size of a block is less than ***MAX_BLOCK_SIZE***
- You may assume that the blocks per set is less than ***MAX_BLOCKS_PER_SET***
- You may assume that the number of sets is less than ***MAX_SETS***
- You may assume that arr has a starting address in set ***0***
- You may use the function which will access the memory at the ***pointer*** passed in.

  int ACCESS(char *)

  It will return 1 on a hit and a 0 on miss in the cache.

a) Complete the following program to calculate the ***block size***. ***(3 pts)***

```
uint64_t MAX_CACHE_SIZE = MAX_SETS * ______________________________;

// this function returns the block size in bytes as an integer
uint64_t getBlockSize() {
     char arr[MAX_CACHE_SIZE];

     // load in the first block
     ACCESS(___________);

     // iterate over one block
     for (uint64_t i = 0; i < ___________ ; i += ___________ ) {
          if(___________) {
               return i;
          }
     }
}
```

b) Now assume that we have calculated the block size and this is stored in the global variable **BLOCK_SIZE**. Complete the following program to determine the **blocks per set.** You should use previously computed values instead of MAX values wherever possible. ***(3 pts)***

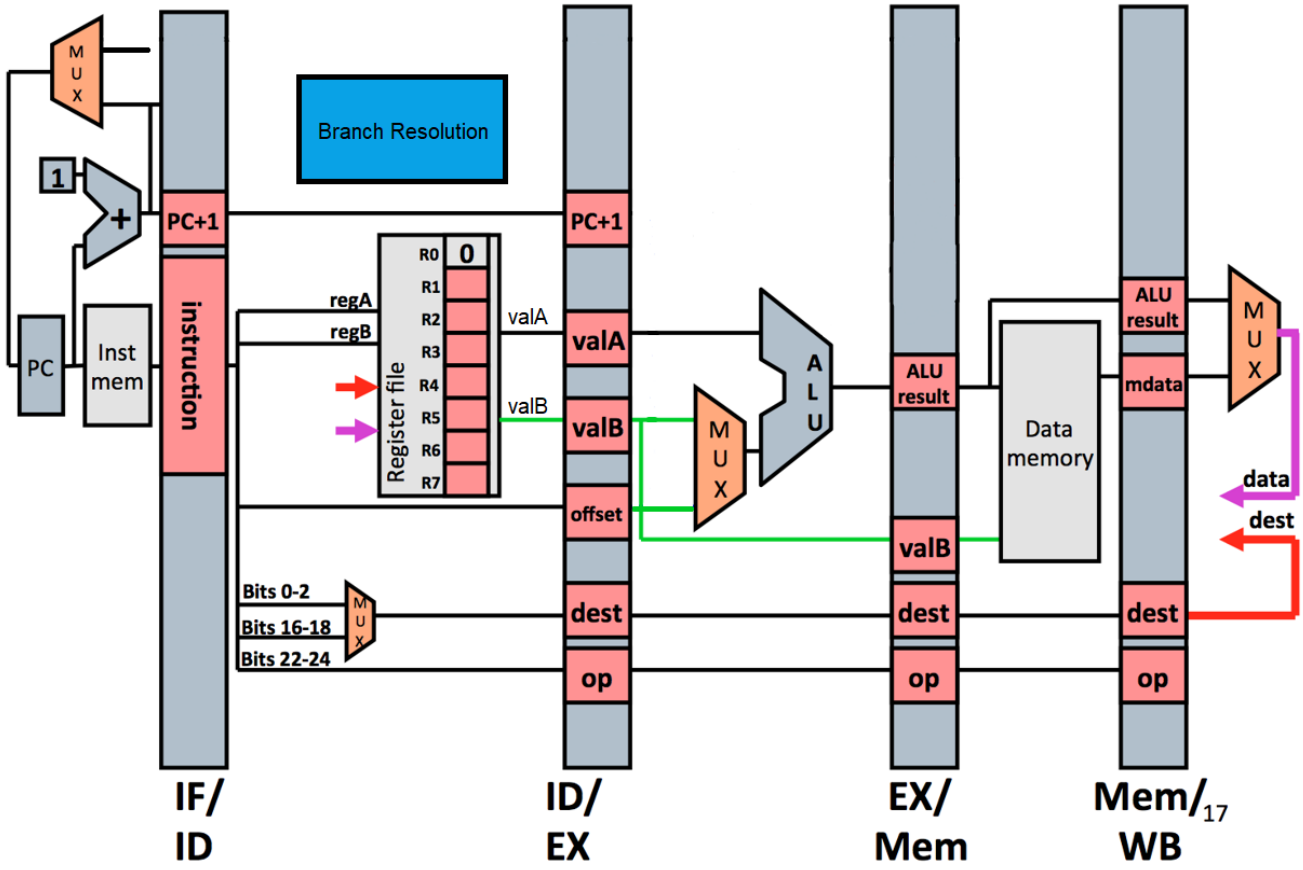
```
uint64_t MAX_CACHE_SIZE = MAX_SETS *  _____________________________;

// this function returns the blocks per set as an integer
uint64_t getBlocksPerSet() {
      char arr[MAX_CACHE_SIZE];

      // incrementing by setStep should always load into the same set
      uint64_t set_step = _____________________________;

      // this loop should increment the number of blocks loaded into a
      // single set until there is an eviction
      for (uint64_t blocks_per_set = 1;
            blocks_per_set <= MAX_BLOCKS_PER_SET;
            blocks_per_set *= 2) {
            for (uint64_t i = 0; i < blocks_per_set *2; i++) {
                  ACCESS(arr + ____________ * set_step );

            }

            if(!ACCESS(arr)) {
                  return ____________________ ;
            }
      }
}
```

c) Now assume that we have calculated the block size and the blocks per set and these are stored in the globals ***BLOCK_SIZE*** *and* ***BLOCKS_PER_SET*** *respectively*. Complete the following program to find the number of sets. You should use previously computed values instead of MAX values wherever possible. ***(4 pts)***

```
uint64_t MAX_CACHE_SIZE = MAX_SETS *  _____________________________;

// this function returns the number of sets as an integer
uint64_t getNumSets() {
      char arr[MAX_CACHE_SIZE];

      // incrementing by set_step should always load into the same set
      uint64_t set_step =   _____________________________;

      // This loop should fill a set and then see if loading some
      // element causes an eviction
      for (uint64_t num_sets = 1; num_sets < MAX_SETS; num_sets *= ______ ) {
            // fill a set

            for (uint64_t j = 0; j <  ___________________ ; j += set_step ) {
                  ACCESS(arr +  ___________________);
            }

            ACCESS(arr + 2 * ___________________ * BLOCK_SIZE);

            if(!ACCESS(arr)) {
                  return ___________________ ;
            }
      }
}
```

Question 7: <u>Hidden Valley Branch Dressing</u> ***{ 8 Points }***

The EECS 370 IAs have become impatient with waiting until the memory stage to resolve branches. They want branches to be resolved in the **decode** stage instead. Below is a pipeline with a block labeled "**Branch Resolution**", this block contains all the logic a pipeline needs to resolve a branch in the decode stage. **All the branch logic has been removed from the EX stage as all branches will be resolved in ID now**.



a. Below are 2 checkboxes full of possible inputs and outputs for the Branch Resolution block. Select the minimum necessary inputs and outputs to make branch resolution possible in the decode stage.
Remark: eq? is an output bit of an ALU that is raised to 1 when the input values are equal, and 0 otherwise ***(3 pts)***

Inputs

- ☐ PC + 1
- ☐ regA
- ☐ regB
- ☐ Offset
- ☐ valA
- ☐ valB

Outputs

- ☐ eq?
- ☐ PC + 1
- ☐ targetPC
- ☐ Offset
- ☐ destReg

b. With this new branch implementation, how many instructions are squashed when a branch is mispredicted? ***(Ex: predict not-taken, but we have to branch)*** ***(2pts)***

Answer:

c. This new design introduces a new kind of data hazard. What is it? Be specific and show an example using LC2K instructions. ***(3pts)***

Question 8: Virtual Memory Reverse Engineering ***{ 12 Points }***

Consider the following virtual memory accesses. You may assume that:

- No other processes are running in this system.
- There were no other memory accesses before this point.
- The table is ordered from least recent access (top) to most recent (bottom)

Virtual Address	**Page Fault**
0x3B	***Y***
0xA2	***Y***
0x31	***N***
0xBA	***Y***

(a) You are given the following info:

- You are in a **12-bit byte addressable** system
- There are 1024 bytes of physical memory
- Single-level page table
- All sizes are powers of 2
- Page table entries are 2 bytes

(i) What is the page size? ***(2pt)***

Answer:

(ii) What is the size of the page table (in bytes)? ***(2pt)***

Answer:

(iii) Fill in the following table with the number of bits composing the physical address. For example, a potential answer could be "2 bits for PPN, 3 bits for PO". ***(2pts)***

Physical Page Number	***Page Offset***

Consider a virtual memory system with **three-level hierarchical page-table**, TLB, ***virtually addressed*** data-cache, and the following specs:

- TLB Access Time: 1 Cycle
- Data-Cache Access Time: 1 Cycle
- Memory Access Time: 50 Cycles
- Disk Access Time: 90,000 Cycles

NOTE:

- The TLB and cache are **NOT** accessed in parallel
- On a 1st level page fault, creation of a 2nd level page table occurs in parallel with accessing disk. On a 2nd level page fault, creation of a 3rd level page table occurs in parallel with accessing disk.
- The page table cannot be cached and is available in main memory.
- Pages that aren't in memory are on disk.
- Upon retrieval from cache, main memory, or disk, data is sent immediately to the CPU, while other updates occur in parallel.

(b) Provide all possible memory access latencies, **in cycles**, for the following situations. *There may be one or more answers. Include all possible answers. Final answer(s) must be a number, not a numeric equation(s).*

I. When data is in the data-cache? ***(2pt)***

Final Answer(s):

II When data must be fetched from memory? ***(2pt)***

Final Answer(s):

III When data must be fetched from disk? ***(2pt)***

Final Answer(s):

Question 9: It's all Coming Together ***{ 15 Points }***

Recall that a Von Neumann architecture places instructions and data in the same memory. If we consider the pipeline discussed in lecture and project 3, we see that we use separate instruction and data memory, which violates the assumptions of a Von Neumann architecture. Let's create a new pipeline architecture that uses **unified memory**. We will have IF and MEM read from the same, shared memory. The memory system we have decided to use only allows one access to take place at a time, so we have what is called a **structural hazard** when we attempt to use memory from both IF and MEM in the same cycle.

A **structural hazard** is a hazard arising from the idea that 2 pieces of data cannot be in the same place at the same time - we can only have 1. In this case, we will need to stall to prevent a fetch and load or store from using the memory bus at the same time.

a) Stalling begins with a design decision: when we have something fetching in the IF stage (which in the 370 pipeline, takes place every cycle) and an lw/sw in the MEM stage, we must choose which of these two stages is allowed to access shared memory on that cycle.

   For this problem, we will prioritize shared memory accesses from the MEM stage. In one sentence, explain what would happen if we instead chose to allow fetches from the IF stage to have priority over MEM stage accesses: ***{ 3 Points }***

b) Consider the modified project 3 & 4 starter code excerpt appended in the reference sheet. Write C code for the entire **IF stage**, making sure to handle the stall due to the structural hazard. Assume that data hazard stalls and branch mispredict squashes are handled in the ID and MEM stages, respectively. Also assume the other stages are correct Project 3 implementations. ***{ 4.5 Points }***

```
if (_____________________________________) {
      _____________________________________
      _____________________________________
      _____________________________________
      _____________________________________
} else {
      _____________________________________
      _____________________________________
      _____________________________________
      _____________________________________
}
```

c) Unfortunately, there is still a problem that can occur with the unified memory. Write a short (3 lines or less) LC2K program that will produce different results (register values, memory values, and/or number of instructions run until termination) on the unified memory pipeline and the single cycle processor. ***{ 2.5 Points }***

_____

_____

_____

d) We can introduce more code to fix this bug in the beginning of the MEM stage. As before, assume the rest of the MEM stage is a correct implementation from project 3, modified to use the mem_access function. Write the C code necessary to fix the bug. Incur no more than 3 extra cycles per instance like that in part c.

{ 5 Points, 1 of which will be awarded for incurring the fewest required stalls. }

SHARED MEMORY INTERFACE EXCERPT:

This is similar to project 4, but not found in the starter code:

```
/* Access hidden shared memory, which is a static array defined in
 * another file, and not accessible otherwise.
 * This function should only be called when absolutely necessary.
 * addr is a 16-bit LC2K word address.
 * - write_flag is 0 for reads (fetch/lw) and 1 for writes (sw).
 * - write_data is a word, and is only valid if write_flag is 1.
 * The return value is undefined if write_flag is 1,
 *    and is a word in memory at address addr otherwise.
 * This increments the hidden num_mem_accesses variable each time.
 * Note that if a structural hazard is not respected,
 *    num_mem_accesses will double increment,
 *    and produce an incorrect value.
 */
extern int mem_access(int addr, int write_flag, int write_data);

/* This function reads the hidden variable num_mem_accesses,
 * which is a static variable in another file, and not accessible
 * otherwise. This variable is written by mem_access() only.
 */
extern int get_num_mem_accesses();
```

PROJECT 3 EXCERPT:

Only one line in the state typedef (underlined, italicized, and bolded) has been modified:

```
typedef struct IFIDStruct {
     int instr;
     int pcPlus1;
} IFIDType;

typedef struct IDEXStruct {
     int instr;
     int pcPlus1;
     int readRegA;
     int readRegB;
     int offset;
} IDEXType;
```

EXCERPT CONTINUED ON NEXT PAGE

PROJECT 3 EXCERPT CONTINUED:

```
typedef struct EXMEMStruct {
      int instr;
      int branchTarget;
      int aluResult;
      int readRegB;
} EXMEMType;

typedef struct MEMWBStruct {
      int instr;
      int writeData;
} MEMWBType;

typedef struct WBENDStruct {
      int instr;
      int writeData;
} WBENDType;

typedef struct {
      int pc;
      // memory has been removed to another file as a static variable
      int reg[8];
      int numMemory;
      IFIDType  IFID;
      IDEXType  IDEX;
      EXMEMType EXMEM;
      MEMWBType MEMWB;
      WBENDType WBEND;
      int cycles; /* number of cycles run so far */
} State;

State state;
State newState;
```

EXCERPT CONTINUED ON NEXT PAGE

PROJECT 3 EXCERPT CONTINUED:

```
#define ADD  0
#define NOR  1
#define LW   2
#define SW   3
#define BEQ  4
#define JALR 5
#define HALT 6
#define NOOP 7
#define NOOPINSTR 0x1c00000

// The following functions are correctly defined for you

int field0(int instruction) {
    return( (instruction>>19) & 0x7);
}

int field1(int instruction) {
    return( (instruction>>16) & 0x7);
}

int field2(int instruction) {
    return(instruction & 0xFFFF);
}

int opcode(int instruction) {
    return(instruction>>22);
}

// Copied from project 1
int convertNum(int num) {
    /* convert a 16-bit number into a 32-bit Linux integer */
    if (num & (1<<15) ) {
        num -= (1<<16);
    }
    return(num);
}
```

END PROJECT EXCERPTS