**CSCI 540 Assignment 3 Poojitha Chennuru**

**CWID-50307728**

1. **What do MIR, MPC, MAR, and MDR do in Mic-1 microarchitecture? (10%)**

**Answer:**

Machine can communicate with memory in two different ways:

1. 32-bit word addressable memory port
2. An 8-bit, byte-addressable memory port

32-bit port is controlled by 2 registers **MAR** and **MDR**

**MAR (Memory address register):** MAR is a 32bit address register. MAR contains word addresses, so that the values 0, 1, 2, etc. refer to consecutive words.

**MDR (Memory Data Register):** MDR is data register. When a word address is put in MAR the data is read from that address location and written in to MDR

**MIR (Microinstruction register):** Its function is to hold the current microinstruction,

whose bits drive the control signals that operate the data path.

**MIR Register:**

Icon

Description automatically generated

Addr – Contains the address of a potential next microinstruction.

JAM – Determines how the next microinstruction is selected.

ALU – ALU and shifter functions.

C – Selects which registers are written from the C bus.

Mem – Memory functions.

B – Selects the B bus source.

**MPC (Micro program counter):** MPC contains the next instruction in the program.

When instruction execution is finished MAR is loaded with MPC value.

MPC will update its value with address of next instruction to be executed.

1. **What do A Bus, B Bus, and C Bus do in Mic-2 microarchitecture? (5%)**

**Answer:**

Mic-2 is a 3-bus architecture with two operand buses A and B and C bus is a output bus the result from Alu driver through shifter and put on to C bus and written in to registers.

With implementation of two operand buses the is no need of h register as both operands can be fetched in same CPU cycle.

1. **Where is the cache physically located? What kinds of localities do cache rely on? Briefly explain the localities. (10%)**

**Answer:**

Cache holds the most recently used memory words in a small, fast memory,

speeding up access to them. One of the most effective ways to improve both bandwidth and latency is to use multiple caches. A basic technique that works very effectively is to introduce a separate cache for instructions and data. There are several benefits from having separate caches for instructions and data, often called a split cache.

There mainly three levels of ( L-1,L-2,L-3) cache.

**L1**:- It is reside on CPU chip and it is fastest of all cache.

**L2:**-It may reside between L1 and main memory. It is not on CPU but near to CPU in CPU package connected with high band.

**L3**:- It is on the mother board it is faster than DRAM(MAIN MEMORY)

**Caches depend on two kinds of address locality**:

1. Temporary Locality
2. Spatial Locality

**Spatial locality** is the observation that memory locations with addresses numerically similar to a recently accessed memory location are likely to be accessed in the near future.

**Temporary locality** occurs when recently accessed memory locations are accessed again.

1. **What is a cache line? What is a cache hit? What is a cache miss? What will happen when there is a cache miss? (10%)**

**Answer:**

The main memory divided in to fixed size blocks called **cache lines**.

cache line typically consists of 4 to 64 consecutive bytes. Lines are numbered consecutively starting at 0, so with a 32-byte line size, line 0 is bytes 0 to 31, line 1 is bytes 32 to 63, and so on.

When the CPU produces a memory address, the hardware extracts the 11 LINE bits from the address and uses these to index into the cache to find one of the 2048 entries. If that entry is valid, the TAG field of the memory address and the Tag field in cache entry are compared. If they agree, the cache entry holds the word being requested, a situation called a **cache hit.**

If the needed entry is not in cache, it is called **cache miss**.

In this case, the 32-byte cache line is fetched from memory and stored in the cache entry, replacing what was there.

1. **If on the same machine, the instructions can have different lengths, what are the advantage(s)? What are the potential problem(s)? (10%)**

**Answer:**

Yes, we can provide different lenghts of instructions to the same machine. By doing this below are advantages and potential problems,

**Advantages**,

Instrctions given will be more flexible.

We can use great combinations of registers, memory locations and addressing modes.

Addressing can be more flexible, with various combinations of register and memory references plus addressing modes.

**Potential problems**,

There will be increase in complexity of the CPU.

Can create problems in the efficient functioning of the processor.

1. **Convert the following infix notations to postfix notations. (5%)**
2. A×B+C÷D
3. A+B+C+D
4. A+B×(C-D)
5. **Solution: A×B+C÷D**

((A×B)+(C÷D))

(ABx)+(CD÷)

ABxCD÷+

1. **A+B+C+D**

(((A+B)+C)+D)

((AB++C)+D)

AB+C++D

AB+C+D+

1. **A+B×(C-D)**

A+(Bx(C-D))

A+(Bx(CD-))

A+(BCD-x)

ABCD-x+

1. **On a machine, an instruction is always 16 bits long, and there are 16 registers.**
2. **Briefly explain the idea of expanding opcode. (5%)**

**Answer :**

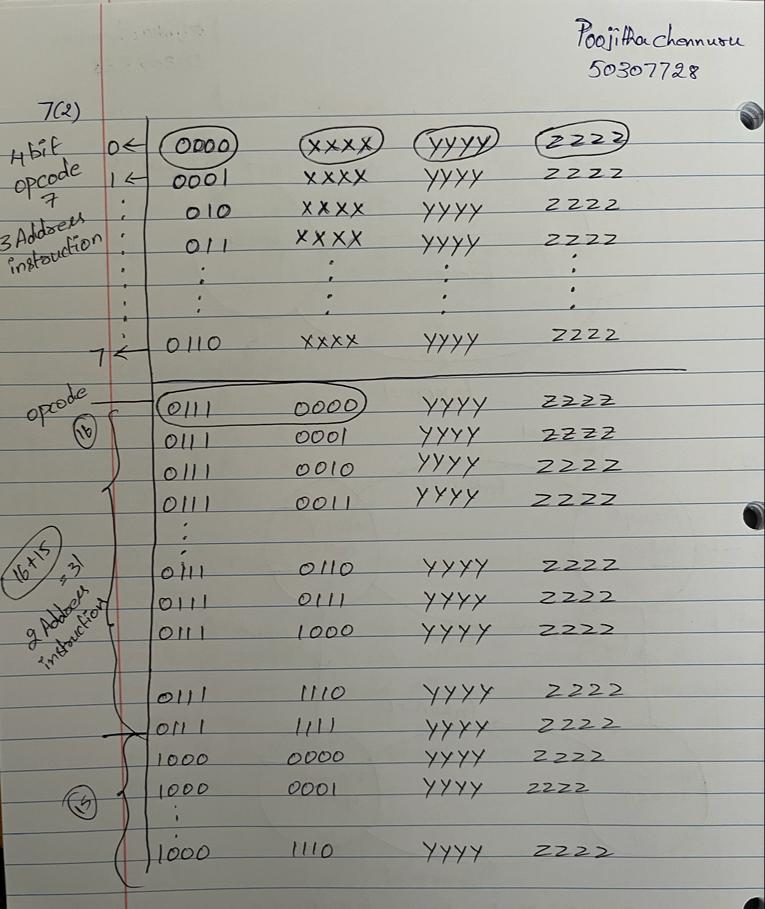
If there are (n+k) bit instruction with a k bit opcode and n bit addresses

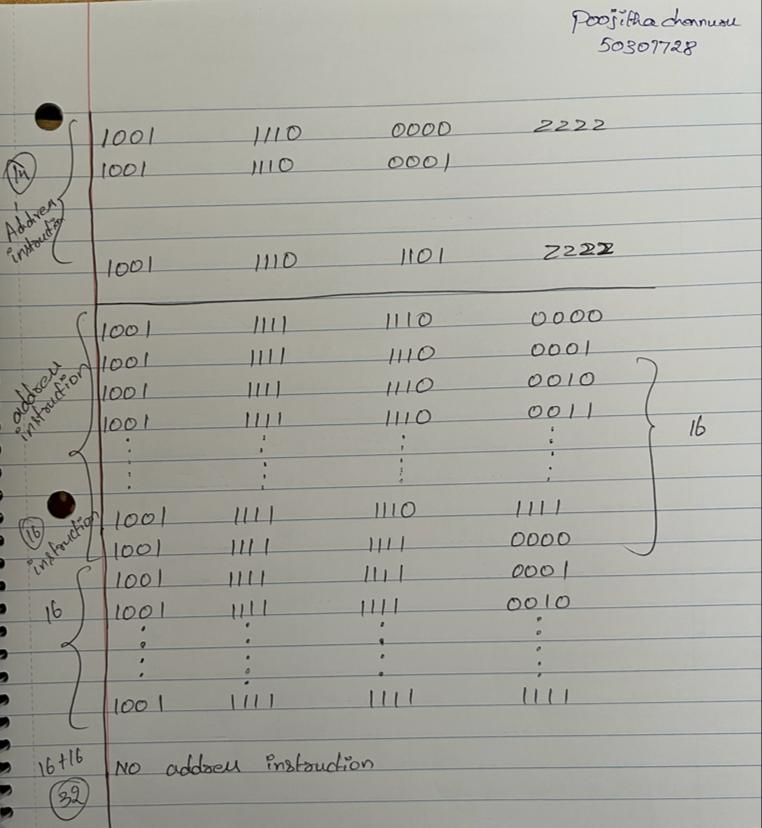
Which means 2k different operations are possible and 2n addressable memory cells. Now without changing instruction length we change the opcode length either increase or decrease i.e. k+1 opcode bits or k-1 opcode bits as instruction length is constant the address bits will become n-1 or n+1 respectively.

By doing this we are adjusting the number of instructions needed at the cost of no of addressable cells. This scheme is called opcode expanding.

1. **Design a scheme to support 7 three-address instructions, 31 two-address instructions, 14 one-address instructions, and 32 zero-address instructions. (10%)**

**Solution:**

****



1. **Assume that the cache size is 256kB, and each cache line is 64 Bytes. (10%)**
2. **Let us assume this is a Two-way associative cache. How many cache sets are there?**

**Solution:** Cache size = 256kB = 2^18 bytes

Size of cache line = 64Bytes = 2^6 bytes

Number of cache lines in the cache = 2^18/2^6 = 2^12 lines.

Two-way associative cache, there will be two lines in each cache set.

Thus, number of cache sets = 2^12/2=2^11 sets = 2048 sets.

Total no of cache lines= 256\*2^10 Bytes/2^6 Bytes=2^12

Total no of cache sets= 2^12/2=2^11

1. Let us assume the following memory blocks need to be accessed:

Memory Blocks #4100, 8196, 2052, 8196, 2052, 4100

If the cache is initially empty, what is the cache hit/miss rate?

**Solution:** Memory Blocks:

#4100, 8196, 2052, 8196, 2052, 4100

Mapping function=N mod S = i

1. 4100 mod 2048 = 4 miss as cache is empty
2. 8196 mod 2048 = 4 miss
3. 2052 mod 2048 = 4 miss (replace 4100 using LRU algorithm)
4. 8196 mod 2048 = 4 hit
5. 2052 mod 2048 = 4 hit
6. 4100 mod 2048 = 4 miss (replace 8196 using LRU algorithm)

Total hits=2

Total misses=4

Hit ratio=2/6=33.33%

Miss ratio =4/6=66.66%

1. Given this piece of code fragment: (10%)

for (int x = 0; x < 10; x++)

{

if ( x % 4 >= 2)

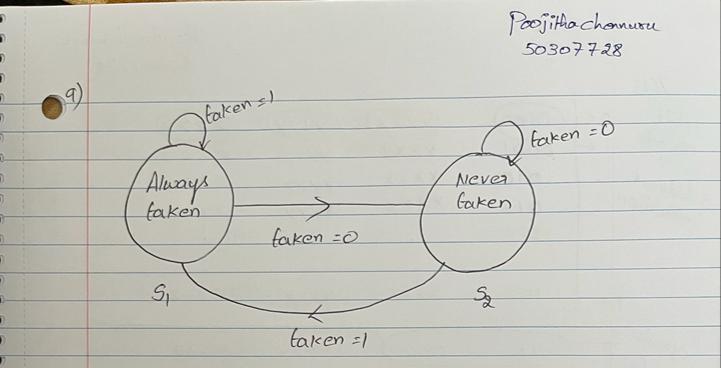
cout << "OK" << endl;

}

Assuming the branch prediction by default is “TAKEN”,

1. What is the accuracy of branch prediction of the **cout statement** when we use a 1-bit branch history?

**Solution:**

****

we have if (x % 4 >= 2) and the values of x are from 0 to 9

for x=0, 0 % 4 =0 and it is not >= 2 so branch won’t take as the default prediction is taken

means we are the taken stage hence it will fail.

Now the prediction changed to branch not taken.

for x=1, 1 % 4 =1 and it is not >= 2 so branch won’t take as we are presently at

not taken stage hence it will pass.

Now the prediction won’t change and it will be still at branch not taken.

for x=2, 2 % 4 =2 and it is >= 2 so branch will take but we are presently at

not taken stage hence it will fail.

Now the prediction will change to branch taken.

for x=3, 3 % 4 = 3 and it is >= 2 so branch will take and we are presently at

taken stage hence it will pass.

Now the prediction won’t change and it will be at branch taken.

for x=4, 4 % 4 = 0 and it is not >= 2 so branch won’t take and we are presently at

taken stage hence it will fail.

Now the prediction will change to branch not taken.

for x=5, 5 % 4 = 1 and it is not >= 2 so branch won’t take and we are presently at

not taken stage hence it will pass.

Now the prediction won’t change and it will be at branch not taken.

for x=6, 6 % 4 = 2 and it is >= 2 so branch will take and we are presently at

not taken stage hence it will fail.

Now the prediction will change to branch taken.

for x=7, 7 % 4 = 3 and it is >= 2 so branch will take and we are presently at

taken stage hence it will pass.

Now the prediction won’t change and it will be at branch taken.

for x=8, 8 % 4 = 0 and it is not >= 2 so branch won’t take and we are presently at

taken stage hence it will fail.

Now the prediction will change and it will be branch not taken.

for x=9, 9 % 4 = 1 and it is not >= 2 so branch won’t take and we are presently at

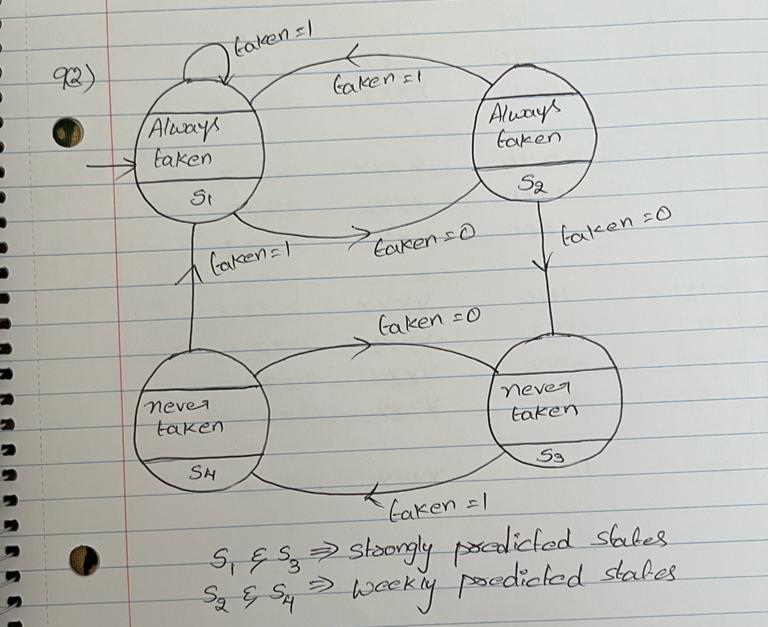
not taken stage hence it will pass.

Now the prediction won’t change and it will be at branch not taken.

Out of 10 cases 5 cases are passed hence our accuracy is 50%.

1. What is the accuracy of branch prediction of the **cout statement** when we use a 2-bit branch history?

**Solution:**

****

S3 and S1 are strongly predicted.

S2 and S4 are weakly predicted.

we have if (x % 4 >= 2) and the values of x are from 0 to 9

for x=0, 0 % 4 =0 and it is not >= 2 so branch won’t take as the default prediction is taken

means we are the taken stage S1 hence it will fail.

Now the prediction changed to branch taken S2.

for x=1, 1 % 4 =1 and it is not >= 2 so branch won’t take as we are presently at

taken stage S2 hence it will fail.

Now the prediction will change to not taken S3.

for x=2, 2 % 4 =2 and it is >= 2 so branch will take but we are presently at

not taken S3 stage hence it will fail.

Now the prediction will change to branch taken S4.

for x=3, 3 % 4 = 3 and it is >= 2 so branch will take and we are presently at

taken S4 stage hence it will fail.

Now the prediction will change and it will be at branch taken S1.

for x=4, 4 % 4 = 0 and it is not >= 2 so branch won’t take and we are presently at

taken S1 stage hence it will fail.

Now the prediction will change to branch taken S2.

for x=5, 5 % 4 = 1 and it is not >= 2 so branch won’t take and we are presently at

taken S2 stage hence it will fail.

Now the prediction will change, and it will be at branch not taken S3.

for x=6, 6 % 4 = 2 and it is >= 2 so branch will take and we are presently at

not taken S3 stage hence it will fail.

Now the prediction will change to branch not taken S4.

for x=7, 7 % 4 = 3 and it is >= 2 so branch will take, and we are presently at

not taken S4 stage hence it will fail.

Now the prediction will change and it will be at branch taken S1.

for x=8, 8 % 4 = 0 and it is not >= 2 so branch won’t take and we are presently at

taken S1 stage hence it will fail.

Now the prediction will change and it will be branch taken S1.

for x=9, 9 % 4 = 1 and it is not >= 2 so branch won’t take and we are presently at

taken S2 stage hence it will fail.

Now the prediction won’t change and it will be at branch not taken S3.

Out of 10 cases 0 cases are passed hence our accuracy is 0%.

1. **Assume we have 8 registers, R0~R7, and we have a pipeline of 6 stages:**

**Instruction Fetch (IF), Instruction Issue (II), Operands Fetch (OF), Execution (EX), Write Back (WB), and Commitment (CO). Each stage needs exactly 1 cycle to finish its work.**

**Also assume that the pipeline supports forwarding, which means the result of WB can be forwarded to OF.**

**Given the following piece of instructions:**

**R1 = R0 + R2**

**R3 = R1 + R4**

**R1 = R5 – R6**

**R5 = R0 + R7**

1. **Identify all the data dependencies and their types. (5%)**

**Solution:**

In the given instructions, we are using R1 which is written recently, so RAW – read after write is applicable.

Then, R1 is writing in the third instruction so, WAR – Write after read is applicable here.

R1 is given the address renaming 2 times, so WAW – write after write applicable here.

So, 3 data dependencies possible are,

**RAW – read after write**

**WAR – Write after read**

**WAW – write after write**

* Now, I1-I2 is a RAW dependency since R1 is the first given value & then it is called in I2.
* I3-I4 is WAR dependency since R5 is called in I3 and then assigned a value in I4.
* I1-I3 is WAW since R1 is given the address renaming 2 times

1. How many cycles do we need if we run the instructions in a 6-stage pipeline that does not support forwarding, register renaming, and out-of-order execution? (II stage only decodes the instruction but does not rename its registers.) (5%)

(3). How many cycles do we need if we run the instructions in the 6-stage pipeline that supports forwarding, register renaming, and out-of-order execution? (5%)

**Solution for 10.2 and 10.3**

Given Instructions,

I1 = R1 = R0 + R2 I1 : ADD R1,R0,R2

I2 = R3 = R1 + R4 I2 : ADD R3,R1,R4

I3 = R1 = R5 – R6 I3 : SUB R1,R5,R6

I4 = R5 = R0 + R7 I4 : ADD R5,R0,R7

If we need to run the given instructions that does not support forwarding, register renaming, and out-of-order execution then, we need to neglect the RAW – read after write and WAW – write after write dependencies.

From below, pipelined execution, Now if we neglect RAW and WAW then **we need 7 cycles to complete the given instructions.**

If we need to run the given instructions that support forwarding, register renaming, and out-of-order then, we just need to draw a table with pipelined execution.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instruction | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 |
| I1 | IF | II | OF | EX | WB | CO |  |  |  |
| I2 |  | IF | II | OF | EX | WB | CO |  |  |
| I3 |  |  | IF | II | OF | EX | WB | CO |  |
| I4 |  |  |  | IF | II | OF | EX | WB | CO |