Name: Đào Trung Hiếu

ID: 23BA14107

Class: C1

\*PRACTICE 1:

Ex 1:

|  |  |  |  |
| --- | --- | --- | --- |
|  | P1 | P2 | P3 |
| Clock rate | 3 GHz | 2.5 GHz | 4 GHz |
| CPI | 1.5 | 1 | 2.2 |

We have formula: CPU Time = (Instruction Count x CPI) / Clock Rate

a)

|  |  |  |  |
| --- | --- | --- | --- |
|  | P1 | P2 | P3 |
| Instruction per second | CPU time x 2x10^9 | CPU time x 2.5x10^9 | CPU time x 1.82x10^9 |

🡪 P2 has the highest performance

b) CPU time = 10s

Clock Cycles = Instruction Count x CPI

|  |  |  |  |
| --- | --- | --- | --- |
|  | P1 | P2 | P3 |
| Number of Instructions | 2x10^10 | 2.5x10^10 | 1.82x10^10 |
| Number of Cycles | 3.10^10 | 2.5x10^10 | 4.10^10 |

c) CPU time new = CPU time old – 30%xCPU time old = 0.7CPU time old

CPI new = CPI old + 20%CPI old = 1.2CPI old

🡪Clock Rate new = (Instruction Count x CPI new) / CPU time new

=(Instruction Count x 1.2xCPI old) / 0.7xCPU time old

=(Instruction Count x 1.2xCPI old x Clock Rate old) / (0.7 x Instruction x CPI old)

= (1.2/0.7) Clock Rate old

≈ 1.714 x Clock Rate old

Ex 2:

|  |  |  |
| --- | --- | --- |
|  | P1 | P2 |
| Clock Rate | 2.5 GHz | 3 GHz |
| CPIs | 1,2,3,3 | 2,2,2,2 |
| Class | A(10%),B(20%),C(50%),D(20%) | A(10%),B(20%),C(50%),D(20%) |

Formula : Global CPI =

a)

|  |  |  |
| --- | --- | --- |
|  | P1 | P2 |
| Global CPI | 1 x 10% + 2 x 20% + 3 x 50% + 3 x 20% = 2.6 | 2 x 100% = 2 |

b)Formula: Clock Cycles = Instruction Count x CPI

CPU Time = Clock Cycle / Clock Rate

1.0E6 = 1x10^6

Instruction Count = 100% x 1x10^6 = 1x10^6

|  |  |  |
| --- | --- | --- |
|  | P1 | P2 |
| Clock Cycles | 2.6x10^6 | 2x10^6 |
| CPU time | 1.04x10^-3 | 6.667x10^-4 |

* P2 is faster than P1

Ex 3:

|  |  |  |
| --- | --- | --- |
|  | A | B |
| IC | 1x10^9 | 1.2x10^9 |
| CPU time | 1.1 | 1.5 |

a)CCT = 1x10^-9

Formula: CPU Time = Instruction Count x CPI x Clock Cycle Time

|  |  |  |
| --- | --- | --- |
|  | A | B |
| CPI | 1.1 | 1.25 |

b)Formula:

Since CPU time are the same 🡪 🡪 🡪

c)New complier: C

|  |  |
| --- | --- |
|  | C |
| IC | 6.10^8 |
| CPI | 1.1 |
| CPU TIme | 0.66 |

🡪

🡪C is slower 0.6 and 0.44 times than A and B respectively

Ex 4:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Arithmetic(A) | Load/Store(LS) | Branch(B) |
| IC | 2.56x10^9 | 1.28x10^9 | 0.256x10^9 |
| CPI | 1 | 12 | 5 |
| CR | 2x10^9 | 2x10^9 | 2x10^9 |

IC per processor of A and LS = IC/0.7p

a)

|  |  |
| --- | --- |
|  | Total CPUT |
| P=1 | 13.425 |
| P=2 | 7.04 |
| P=4 | 3.84 |
| P=8 | 2.24 |

🡪

b)CPI new of A = 2

|  |  |
| --- | --- |
| p | Total CPUT |
| 1 | 15.269 |
| 2 | 7.954 |
| 4 | 4.297 |
| 8 | 2.469 |

c)Decreases CPI of LS to 1p new = 4p old

🡪

🡪

🡪

Ex 5:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| AMD | 4x10^9 | 85% | 700 | 13.7 |

a)Formula:

b)

\*

🡪

Thus, when = , the increase in the CPI is similar to that of the clock rate

c)The CPU Time has been reduced = 825.564 – 700 = 125.564

d)

|  |  |  |  |
| --- | --- | --- | --- |
|  | CPUT | CPI | CR |
| Libquantum(L) | 960x10^-9 | 1.61 | 3x10^9 |

\*

\*

🡪

e)

\*

\*

f)

\*

\*

🡪

Ex 6:

|  |  |  |  |
| --- | --- | --- | --- |
|  | CR | CPI | IC |
| P1 | 4x10^9 | 0.9 | 5x10^9 |
| P2 | 3x10^9 | 0.75 | 1x10^9 |

a)

\*

\*

🡪P2 is faster eventhough CR of P2 is lower

b)

\*Consider 🡪🡪

c)

\*

\*

🡪P1 has higher MIPS eventhough has slower CPUT

d)Formula:

\*

\*

🡪

🡪P1 has higher MFLOPS but IC is higher

Ex 7:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | FP | LS | Branch(B) | Total |
| CPUT | 70 | 85 | 40 | 250 |

🡪CPUT of INT = 55

a)

\*🡪

b)

\*🡪

c)

Since lost 50s, but is only 40s🡪Can’t

Ex 8:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | FP | INT | LS | B |
| IC | 50x10^6 | 110x10^6 | 80x10^6 | 16x10^6 |
| CPI | 1 | 1 | 4 | 2 |
| CR | 2x10^9 | 2x10^9 | 2x10^9 | 2x10^9 |

a)

\*

🡪🡪Can’t improve the CPI of FP instrucƟons if we want the program to run two times faster

b)

\*

🡪🡪Reduces 80% the CPI of L/S instructons if we want the program to run two times faster

c)

\*

🡪

Ex 9:

|  |  |  |
| --- | --- | --- |
| CPUT | Before additional | After additional |
| P=1 | 100 | 100 |
| p |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Actual speedup( | Ideal speedup( |  |
| 2 | 54 | 1.852 | 2 | 0.926 |
| 4 | 29 | 3.448 | 4 | 0.862 |
| 8 | 16.5 | 6.061 | 8 | 0.758 |
| 16 | 10.25 | 9.756 | 16 | 0.61 |
| 32 | 7.125 | 14.035 | 32 | 0.439 |
| 64 | 5.5625 | 17.978 | 64 | 0.281 |
| 128 | 4.78125 | 20.915 | 128 | 0.163 |

Ex 10:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Arithmetic(1) | LS(2) | B(3) |
| I mix | 70% | 10% | 20% |
| CPI | 2 | 6 | 3 |

a)

\*

b)

\*🡪

c)

\*PRACTICE 2:

Ex 1:

a

🡪Binary to decimal

b)

🡪Binary fractional to decimal

c)

\*

\*

\* 🡪

\*

\*

🡪Decimal to binary

d)

\* \*

\* \*

\*

\*

🡪

🡪Decimal fraction to binary

e)

🡪2s-Complement Signed Integers

Ex 2:

a)

\*

\*

\*

\*

\*

\*

\*

🡪(8 bits)

\*Signed Negation🡪

\*

b) Since 🡪

🡪Exponent = 6 + 127 =

|  |  |  |
| --- | --- | --- |
| Sign | Exponent | Fraction(23 bits) |
| 1 | 10000101 | 00010(0…000)🡨18 zeros |

🡪

Ex 3:

\*Underflow < -128 < 127 < Overflow

a)

b)

\*

🡪Magnitude=🡪

\*)

\*

🡪Neither

c)

🡪Underflow

Ex 4:

a)

\*

🡪

\*

🡪

=>

b)

\*

c)

\*

Ex 5:

\* |

\* |🡪🡪Fraction=010101…0101(24 bits)

\* |

🡪This isn’t representation exact

Ex 6:

a)

\*

\*Unsigned integer: Since is positive number 🡪similarly🡪

b)

\*

🡪

|  |  |  |
| --- | --- | --- |
| S | E | F |
| 0 | 00011000= | (00….000)🡨23 bits= |

🡪

c)

\*

\*opcode(7 bits)=0010011🡪I-Format

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| immediate | Rs1 | funct3 | rd | opcode |
|  |  |  |  |  |
| 4 | X11 | addi | X11 |  |

🡪addi rd, rs1, immediate = addi x11, x11, 4

Ex 7:

a)

\*

|  |  |  |
| --- | --- | --- |
| S | E | F |
| 0 |  | 00…00000 |

🡪By IEEE 754 floating-point format:

0 01111101 0000000000000000000000

🡪-0.25=1 01111101 0000000000000000000000=

b)

\*

\*

🡪Are the same

Ex 8:

a)-0.15625

\*

|  |  |  |
| --- | --- | --- |
| S(1bits) | E(5 bits),bias=15 | F(10 bits) |
| 1 | 01100 | 0100000000 |

🡪

b)

This 16-bit floating point format has only 15 bias and 5 bits of exponent but the single precision IEEE 754 standard has 127 bias and 8 bits of exponent

c)

\*

|  |  |  |
| --- | --- | --- |
| S | E | F |
| 0 | 10011 | 1010001000 |

\*

|  |  |  |
| --- | --- | --- |
| S | E | F |
| 0 | 01101 | 1010100100 |

Shift fraction by 6 bits: 0. 0000011010100100

\*1.1010001000000000+0. 0000011010100100=1.1010100010100100

🡪F=1010100010, G=1,R=0,S=1🡪F=1010100010+1=1010100011

|  |  |  |
| --- | --- | --- |
| S | E | F |
| 0 | 10011 | 1010100011 |

🡪

d)

\*

|  |  |  |
| --- | --- | --- |
| S | E | F |
| 1 | 10010 |  |

\*

|  |  |  |
| --- | --- | --- |
| S | E | F |
| 1 | 01100 | 0111000010 |

Shift fraction by 6 bits:0.00000

🡪

🡪

|  |  |  |
| --- | --- | --- |
| S | E | F |
| 0 | 01001 | 0111001100 |

\*G=0,R=0,S has 1🡪S=1🡪F=0111001100

🡪

Ex 9:

a)🡪

b)

🡪

|  |  |  |
| --- | --- | --- |
| S | E | F |
| 0 |  | 10001001010000000000000 |

🡪010001001010000000000000=

c-i)

c-ii)Since sign = 0🡪positive number 🡪2’s complement is the same

\*PRACTICE 3:

Ex 1:

a)Since on a big-endian machine and the data at address is

🡪: Byte at is

🡪

b) Since on a little-endian machine and the data at address is

🡪: Byte at is

🡪

Ex 2:

a)

🡪

\*Since RISC-V is only 32-bit 🡪

\*Since and 🡪 Overflow

b)

🡪

🡪Not overflow

c)

\*(Overflow)

\*

🡪Overflow

Ex 3:

\*Since is invert and store to

🡪A set of RISC-V instruction:

Ex 4:

\*We have and is the base address

🡪 is mean 🡪An instruction: (C is statement) (1)

\* is means shift left 4 bits

\*Since 🡪shift left 4 bits 🡪 An instruction: (2)

\*Combining (1) and (2) we have a sequence of RISC-V assembly instruction:

Ex 5:

a)

:

b)

\*Call

🡪

🡪

🡪

\*Call

🡪

🡪

🡪

\*Call

🡪

🡪

🡪

Ex 6:

a) We have

\*This RISC-V loop means: comparing with then do and and then back to the loop and continues compare till then go to DONE

🡪 🡪 🡪

b)🡪 The equivalent C code:

;

c)

\*Since the loop has 4 instructions 🡪 Runs instructions (

\*When the RISC-V runs 1 instruction which is compare with and go to DONE

🡪Number RISC-V instructions are executed =

d)Replace to we have new loop:

\*Then this RISC-V changes the meaning is: DONE when 🡪 We have the equivalent C code:

;

Ex 7:

a)We have:

\*Since

\*We call and uses to compare in the loop

\*

means: add 8 bytes to move 1 element in MemArray

🡪 The equivalent C code:

;

;

b)Reduce:

\*If continues loop

\*Original: loop has 5 instructions and at run 5 instructions 🡪 from run 100 times then RISC-V runs

\*Reduce: still run 100 times from but only 1 instruction outside loop then RISC-V runs 501 instruction

Ex 8:

a)Let 🡪

b)We can’t use tail-call because we use the first call g(a,b) and second call g(g(a,b),c+d) which the first call is not the result of f

c)

\*Since

\*

\*

\*

\*

\*PRACTICE 4:

Ex 1:

🡪 R-format

a)Since R-format:

ALU operation type = 10

b) Block:

c)

\*Block produce no output:

\*Block produce output that is not used:

Ex 2:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| R-type | I-type(non-lw) | Load | Store | Branch | Jump |
| 24% | 28% | 25% | 10% | 11% | 2% |

a)

b)

c)

d)

Ex 3:

\*Instruction word:

\*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| imm | Rs2 | Rs1 | Funct3 | imm | opcode |
| 0000000 | 01100 | 01101 | 011 | 10100 | 0100011 |

🡪Instruction:

a) ALU control unit’s inputs:

\*

\*

b)

c)

\*

\*

\*

\*

d)

\*

\*

\*

e)

\*

\*

\*

\*

\*RegWrite=0

Ex 4:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| I-Mem/D-Mem | Register File | Mux | ALU | Adder | Single gate | Register Read | Register  Setup | Sign  extend | Control |
| 250 ps | 150 ps | 25 ps | 200 ps | 150 ps | 5 ps | 30 ps | 20 ps | 50 ps | 50ps |

a)

b)

c)

d)

e)

f)

Ex 5:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| IF | ID | EX | MEM | WB |
| 250ps | 350ps | 150ps | 300ps | 200ps |

|  |  |  |  |
| --- | --- | --- | --- |
| ALU/Logic | Jump/Branch | Load | Store |
| 45% | 20% | 20% | 15% |

a)

\*

\*

b)

\*

\*

c)

\*

d)

\*

e)

\*

Ex 6:

The circuit is divided into k parts by a K stage pipeline. It is K times faster because each stage has the same transistor delay (ideally). As we have k stage pipeline. For the first instruction we take k cycles and for the next n-1 instructions we take only n-1 cycles. So the total clock cycles will be k+(n-1).

Ex 7:

\*Add NOP:

Ex 8:

|  |  |  |
| --- | --- | --- |
|  | Before forwarding | After forwarding |
| Cycle time | 250ps | 300ps |
| Number of NOPs | 0.4n | 0.05n |

a)

\*

b)

\*Since the cycle time is equally🡪🡪

🡪16.67% NOPs remain

c)

\*

d)

\*Since 🡪 Faster after forwarding

e)

\*When NOPs after forwarding NOPs, a program will run slower. So that NOPs at minimum is equal to NOPs

Ex 9:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| R-type | Beqz/bnez | jal | lw | sw |
| 40% | 25% | 5% | 25% | 5% |

|  |  |  |
| --- | --- | --- |
| Always-Taken | Always-Not-Taken | 2-bit |
| 45% | 55% | 85% |

a)

b)

🡪

c)

🡪

d)

\*

🡪

🡪

e)

\*

🡪

\*

🡪

\*Since

🡪

f)

\*Since 80% of all executed branch instructions are always predicted correctly that means the accuracy is 100%, then the accuracy on the remaining 20% of the branch instructions for 2-bit, the accuracy is 85%, is

Ex 10:

a)

\*

\*

b) First 4 branch:

\*

🡪 The accuracy of the 2-bit predictor for the first four branches =

c)

\*

🡪After the sequence ( each 5 prediction have 3 correct then after the sequence the accuracy of the 2-bit predictor is

d) A sequential circuit that each T must predict taken and each NT must predict not taken, also T provides 1 and NT provides 0

e)

\*This means each prediction is always incorrect, then the accuracy = 0%

f) Since after an identity sequence, it will repeat a sequence with 5 predictions. Therefore, we save predictions that can perfectly predict both of pattern in 5-bit shift register. Thus, it will be perfectly predictable

\*PRACTICE 5:

Ex 1:

a)

b)

c)

d)

e)

f) Since

<

\*Since each cache block stores 2 elements🡪

Ex 2:

a)

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |
| 0000 0011 | 1011 0100 | 0010 1011 | 0000 0010 | 1011 1111 | 0101 1000 | 1011 1110 | 0000 1110 | 1011 0101 | 0010 1100 | 1011 1010 | 1111 1101 |

🡪 Tag and index is not the same, then all miss

b)

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |
| 0000 001 1 | 1011 010 0 | 0010 101 1 | 0000 001 0 | 1011 111 1 | 0101 100 0 | 1011 111 0 | 0000 111 0 | 1011 010 1 | 0010 110 0 | 1011 101 0 | 1111 110 1 |

🡪Hit:

Miss: all remaining

c) C1 cache has 8 entries (Cache index 3 bits), C2 4 entries (Cache index 2

bits), C3 2 entries (Cache index 1 bit)

Ex 3:

|  |  |  |
| --- | --- | --- |
| Tag | Index | Offset |
| 63-10 | 9-5 | 4-0 |

a)Offset 5 bits 🡪

b)Index 5 bits 🡪

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | | | |
| Hex | 00 | 04 | 10 | 84 | E8 | A0 | 400 | 1E | 8C | C1C | B4 | 884 |
| Dex | 0 | 4 | 16 | 132 | 232 | 160 | 1024 | 30 | 140 | 3100 | 180 | 2180 |

c)

|  |  |  |  |
| --- | --- | --- | --- |
| Dec | Tag | Index | Offset |
| 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 4 |
| 16 | 0 | 0 | 16 |
| 132 | 0 | 4 | 4 |
| 232 | 0 | 7 | 8 |
| 160 | 0 | 5 | 0 |
| 1024 | 1 | 0 | 0 |
| 30 | 0 | 0 | 30 |
| 140 | 0 | 4 | 12 |
| 3100 | 3 | 3 | 20 |
| 180 | 0 | 5 | 20 |
| 2180 | 2 | 4 | 8 |

🡪Hit:4, 16,140,180

Miss: all remaining

Replace: 1024, 30, 2180

d)

e)

<0, 0, Mem[0x000]–Mem[0x0FF]>

<3, 3, Mem[0xC00]–Mem[0xCFF]>

<4, 2, Mem[0x880]–Mem[0x97F]>

<5, 0, Mem[0x0A0]–Mem[0x19F]>

<7, 0, Mem[0x0E0]–Mem[0x1DF]>

Ex 4:

|  |  |
| --- | --- |
| L1 | L2 |
| Write through, non-write allocate | Write back, write allocate |

a)

\*

\*

b) On an L1 write miss, the word is written directly to L2 without bringing its block into the L1 cache. If this results in an L2 miss, its block must be brought into the L2 cache from Memory, possibly

replacing a dirty block, which must first be written to memory

c) After an L1 write miss, the block will reside in L2 but not in L1. A subsequent read miss on the same block will require that the block in L2 be written back to memory, transferred to L1, and invalidated in L2

Ex 5:

a) If the stream signifies word addresses, the first address (0) will miss and bytes 0–31 will be brought in, making the accesses to 1, 2, 3, 4, 5, 6, 7 hits. Then 8 will miss and bytes 32–63 will be brought in... So the miss rate will be 1/8. The misses are compulsory and constructed only on the access pattern and the block size.

b)

\*

\*

\*

🡪 Spatial locality

c) The first access is a compulsory miss and all subsequent accesses are found in the stream buffer 🡪 The miss rate = where N is the number of memory accessed =

Ex 6:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Dec | 4669 | 2227 | 13916 | 34587 | 48870 | 12608 | 49225 |
| Hex |  |  |  |  |  |  |  |

TLB

|  |  |  |  |
| --- | --- | --- | --- |
| Valid | Tag | Physical page number | Time since last access |
| 1 |  | 12 | 4 |
| 1 |  | 4 | 1 |
| 1 |  | 6 | 3 |
| 0 |  | 9 | 7 |

Page table

|  |  |  |
| --- | --- | --- |
| Index | Valid | Physical page or in disk |
| 0 | 1 | 5 |
| 1 | 0 | Disk |
| 2 | 0 | Disk |
| 3 | 1 | 6 |
| 4 | 1 | 9 |
| 5 | 1 | 11 |
| 6 | 0 | Disk |
| 7 | 1 | 4 |
| 8 | 0 | Disk |
| 9 | 0 | Disk |
| a | 1 | 3 |
| b | 1 | 12 |

a-i) Miss: all

a-ii) Miss: all

a-iii) Page faults: all

a-iv)

|  |  |  |  |
| --- | --- | --- | --- |
| Valid | Tag | Physical page number | Time since last access |
| 1 |  | 12 | 4 |
| 1 |  | 4 | 1 |
| 1 |  | 6 | 3 |
| 1 |  | 9 | 7 |

b-i) Hit:

Miss:

b-ii) Hit:

Miss:

b-iii) Page faults:

b-iv)

|  |  |  |  |
| --- | --- | --- | --- |
| Valid | Tag | Physical page number | Time since last access |
| 1 |  | 5 | 4 |
| 1 |  | 4 | 1 |
| 1 |  | 6 | 3 |
| 1 |  | 9 | 7 |

🡪 Advantages: smaller page tables, fewer TLB misses, reduced page table overhead, better I/O efficiency

Disadvantages: increased internal fragmentation, more data transferred on page faults, worse cache behavior, less flexibility in memory allocation

c-i) Hit:

Miss: all remaining

c-ii) Hit:

Miss:

c-iii) Page fault:

c-iv)

|  |  |  |  |
| --- | --- | --- | --- |
| Valid | Tag | Physical page number | Time since last access |
| 1 |  | 5 | 4 |
| 1 |  | 4 | 1 |
| 1 |  | 6 | 3 |
| 1 |  | 9 | 7 |