1. Definitions
   1. RISC: reduced instruction set computer. It allows a computer’s microprocessor to have fewer cycles per instruction (CPI) than a complex instruction set computer (CISC)
   2. CISC: complex instruction set computer. A computer in which single instructions can execute several low-level operations (such as a load from memory, an arithmetic operation, and a memory store) or are capable of multi-step operations or addressing modes within single instructions.
   3. Architecture: Architecture describes the design, structure, and logic of computer systems.
   4. MIPS Instruction Type: classification of MIPS instructions according to coding format. 4 categories: R-type, I-type, J-type, and coprocessor.
   5. Harvard Architecture: stores machine instruction and data in separate memory units and are connected by different (signal) busses.
   6. Von Neumann Architecture: stores data and machine instruction in the same memory and is connected to the rest of the computing units by a single data bus.
   7. Multicore: a single processor chip contains multiple cores working together as compared to a single-core processor in CPU architecture.
   8. DMA: Direct Memory Access. Method of transferring data from memory to other components of the computer without any interference of the CPU.
   9. L1 cache: CPU cache in a multi-cache architecture. Smallest in size and fastest as well as closest to the CPU core.
   10. Cache Coherence: Concern of regularity of data stored in cache memory. Holds the importance of data being the same in all core caches.
2. MIPS Instruction Type: classification of MIPS instructions according to coding format. 4 categories: R-type, I-type, J-type, and coprocessor.
   1. R-type: uses opcode of 0, and differentiated by their functional values. Only uses registers past the first three shift instructions. Allows for jumps and other syscall.
   2. J-type: uses opcode of 2 and 3. Uses pseudo-absolute addressing for jumps.
   3. I-type: uses opcode greater than 3. Instructions have 16-bit immediate that sign-extends to a 32-bit value in every instructions. Branch instructions multiply immediate by 4.
3. Given:

0x1004 bne R1, R2, 0x0e04

0x1008 bgez R3, 0x1098

Converted:

000101 00001 00010 111111110111111 (how: bne: 0x05, r1: 0x01, r2: 0x02, BFC = PC + 4 + 4\*BFC, 0x1004+4+4\*BFC= 0x0e04, BFC = 0xFF7f (-12910)

00001 00011 00001 0000000000100011 (how: sim steps above so, 0x1008+4+4\*BFC = 0x1098, BFC=0x023 (3510)

1. Memory describe memory hierarchy for cloud storage, registers, L1 cache, main memory, L2 cache, L3 cache and mass storage
   1. The memory hierarchy is as following:
      1. level 0: for registers.
         1. inside CPU, least access time, smaller in size and most expensive
      2. level 1: for L1, L2, L3 cache.
         1. L1 is primary cache, L2 is secondary cache and L3 is specialized memory to improve performance of L1 and L2
      3. level 2: Main memory (RAM)
         1. less expensive than cache and larger in size
      4. level 3: online Mass storage, secondary storage
      5. level 4: offline bulk storage, cloud storage
         1. used to store removable files
   2. access times increases as we move from level 4 to level 0. Therefor:
      1. registers: fastest access time = 1 time unit
      2. cache: 6KB-2MB in size, 30 BG/s
      3. main memory: GB in size, best access speed, 10 GB/s
      4. disk storage(mass): TB in size, access speed- 2000 MB/s
      5. tertiary storage (cloud): up to exabytes in size, best access speed – 160 MB/s
   3. C
2. A & B) A screenshot of a cell phone

   Description automatically generatedA screenshot of a cell phone

   Description automatically generated

A screenshot of a computer

Description automatically generated

C) MIPS is attached via email

1. Architecture
   1. CPI = % of each instruction \* # of cycles each takes

CPI = (.6\*5)+(.1\*4)+(.15\*4)+(.06\*3)+(.9\*3)

CPI = 3.0 + 0.4 + 0.60 + 0.18 + 2.7 = 6.88

* 1. CPI2 = (0.6\*5)+(0.1\*4)+(0.15\*4)+(0.06\*3)+(0.9(0.75(3)+0.25(5)))

CPI2 = 3+.4+.6+.18+.9(3.5) = 7.33

* 1. The cycles required in CPI2 is greater than the cycles required in CPI
  2. CPI3 = 0.9(1)+0.1(12) = 2.1

1. “data” buffer overflow in von Neumann machine allows an attacker to gain data after the buffer (variables, return pointers and frames) because it starts in high and ends in low address memory. While Harvard architectures has a stack that starts in low and grows to high address memory.
2. Explain MIPS
   1. $4 = ~$4

$4 = $4 & $12

Contents of $4 stored into $5 … $5 = $4

$3 = $3 + $2

* 1. $5 = $3 >> $5

$5 = $5 << $2

1. Tagged architecture allows memory to be safe and protected by allowing everything to be tagged. This allows data to be revealed only to those that were given permission by the tagging arithmetic.
2. i/o port
   1. 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
   2. 4, 5, 6, 7, 8, 9, 10, 11 are outputs and the rest are inputs
   3. $4 = 0000 0000 0000 0000 0000 0000 1111 1111

$5 = 0000 0000 0000 0000 0000 0000 0011 1100

$6 = 0000 0000 0000 0000 0000 0000 1010 0101

The value of the input bits is 0x81

1. $4 = 0000 0000 0000 0000 0000 0000 1111 1111

$5 = 0000 0000 0000 0000 0000 0000 0011 1100

$6 = 0000 0000 0000 0000 0000 0000 0111 1111

The value of the output bits is 0x3C

* 1. Without understanding C code, you wouldn’t understand the purpose of the code. Without understanding the code, you wouldn’t know how to convert it to MIPS. But knowing how the code operated allowed you to understand and see how the converted code is run and operated in machine code with registers and memory.
  2. Whenever a signals crosses a set limit, the pulse’s value was changed. When the pulse increased to the new pulses for each limit then it was considered a clean pulse.

.global \_start

addi $1, $0, 0x100 # in[]

addi $2, $0, 0 # i

addi $3, $0, 0 # cnt

addi $4, $0, 0 # en

addi $5, $0, 1 # (cnt > 1)

addi $6, $0, 0 # in\_old

addi $7, $0, 5\*4

loop:

beq $2, $7, exit

lw $9, 0($1)

beq $8, $0, if

bgt $3, 1, en

add $6, $0, $9

addi $9, $9, 4

addi $2, $2, 4

j loop

if:

bgt $1, 10, add

bgt $3, 1, en

add $6, $0, $9

addi $9, $9, 4

addi $2, $2, 4

j loop

add:

addi $3, $3, 1

bgt $3, 1, en

add $6, $0, $9

addi $9, $9, 4

addi $2, $2, 4

j loop

en:

addi $4, $0, 1

add $6, $0, $9

addi $9, $9, 4

addi $2, $2, 4

j loop

exit:

nop

* 1. Combining registers together that hold variables
  2. A lot
  3. A lot lol