CA

Q-1 spring – 2022

~~set~~ – A

① 1000 1101 0010 1000 0000 0000 0011 1000

1000 1101 0010 1000, 0000 0000 0011 1000

opcode = 35 thus I type. and 'lw' instruction.

rs = 9 so register (source) = \$t1

rt = 8 so " (target) = \$t0

offset = 56.

so, instruction in MIPs : lw \$t0, 56(\$t1)

~~set~~

② Activities of PC register (Programme Counter):

1. Instruction Fetch: PC holds the memory address of the next instruction to be executed. During the fetch stage of the instruction cycle, the PC sends this address to RAM and the instruction at that address is fetched into the instruction cache or the CPU's instruction pipeline.

~~2. Increment:~~

2. Branching: In Branch instr. or jumps, the PC is updated to a new address specified by the instruction. This allows the CPU to change the flow of execution by jumping to a different part of the program or subroutine based on a condition or an unconditional jump.

3. Exception Handling: Exceptional events like interrupts, traps or system calls, the PC can be updated to the address of an exception handler routine. This redirection of the PC allows the CPU to handle these events and later return to the normal program flow.

4. Function Calls and Returns: when a f^n is called PC may be updated to the address of the function's entry point. After executing the f^n, a return instr. updates the PC with the address to resume execution at the point ~~after~~ the f^n call.

5. Sequential Execution: In the absence of branches or jumps, the PC simply continues to increment, ensuring that the instr. are executed and feted properly.

6. Pipelining: In modern CPUs with instr. pipelines, multiple stages of the pipeline may have seperate PC values, each corresponding to a different instr. in the pipeline. This allows for concurrent execution of multiple instr.

set B

① lw $t2 , 72 ($s1) .

I type, so opcode = 100011 (35 for lw) .

$t2 is target register no. 10 so rt = 01010

$s1 is source register no. 17 so rs = 10001

and offset is 72 so offset = 0000 0000 0100 1000

So, machine codes:

op	rs	rt	offset
100011	10001	01010	0000 0000 0100 1000

(lw)

② Jump or J instruction has the greatest range among the mention instructions. We know, for j instr

op	target address.
6 bits	26 bits.

So, they allows 26 bit immediate value which concates with the upper 4 bits of the PC registers (PC+4) to form 32 bits. target address and so they can jump any address within 2^{26} or 64 MB.

Incase of branch instructions, they are I-type, that means

op	rs	rt	offset
6 bits	5 bits	5 bits	16 bits

they allow 16 bit immediate value that gets added to the PC if the specified condition is met. This means they have a relatively small range and incase of jr instruct.

it is used to jumpto an address ~~range~~. stored in a register. while it allows for indirect branching, it doesn't have the same long range capability as j instr. The range is Limited to 32 bit.

set C

⊛ Given, $\underbrace{0001\ 00}\underbrace{01\ 000}\underbrace{0\ 1001},\underbrace{0000\ 0000\ 0011\ 1001},$

I-type as opcode = 4 which means 'beq'
rs = 8 which means $t0
rt = 9 " " $t1

and immediate value = 57

so, beq $t1, $t0, 57.

⊛ Endcoding the 32 bit dest. address in j Type instruction :

The format of j type is,

opcode	~~immediate~~ target address
6 bits	26 bits

MIPS jump j instr. replaces lower 28 bits of the PC with A00 where A is the 26 bit address; it never changes upper 4 bits.

Exple: if PC = 1011X (where X = 28 bits) then after replacement it will be 1011A00.

There are 16 ($=2^4$) partitions of the 2^{32} size address space, each partition of size-256 MB ($=2^{28}$) such that, in each partition the upper 4 bits of the address is same.

Incase a program crosses an address partition, then a j that reaches a different partition has to be replaced by jr with a full 32-bit address first loaded into the jump register.
Therefore, OS always try to load a program inside a single partition.

set-D

① sw $t3, 72($s4).

Here, it is I-type.
opcode = 43 since 'sw'.
rs = 20 since $s4
rt = 11 " $t3.
offset = 72 which means. 0000 0000 0100 1000

so,

opcode	rs	rt	immidiate address value
101011	10100	01011	0000 0000 0100 1000

Thus, 101011 10100 01011 0000 0000 0100 1000.

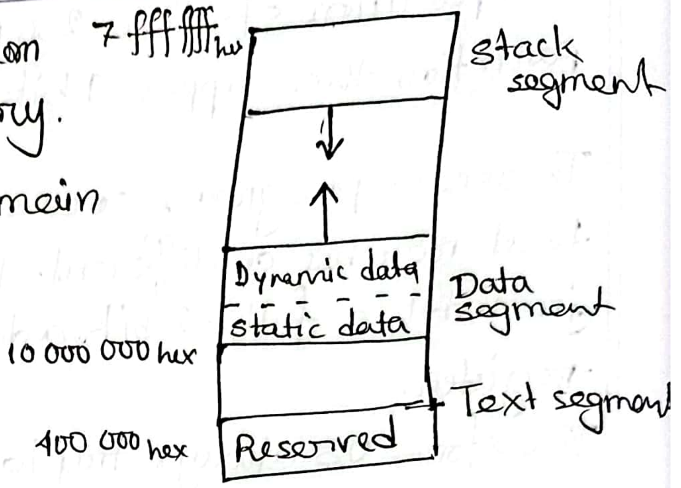
② Uses of Loader program :

1. Reads the executable file header to determine size of the text and data segments.
2. Creates an address space large enough for the text and data.

3. Copies the instr. and data from the executable file into memory.

4. Copies the parameters to the main program onto the stack.

5. Initializes the machine registers and sets the stack pointer to the first free location.



6. Jumps to a start-up routine that copies the parameters into the argument registers and calls the main routine of the program. When the main routine returns, the start-up routine terminates the program with an exit system call.

Set - E

(1). $\underbrace{1000\ 11}\underbrace{10\ 011}\underbrace{0\ 1000}\underbrace{0000\ 0000\ 0010\ 1000}$

opcode = 35 , so it's I-type, 'lw'

rs = 19, so. register is $\$s3$

rt = 8, so " " $\$t1$

offset = 40 .

So, lw $\$t1$, 40($\$s3$).

(2) Format of branch instruction is,

opcode	rs	rt	offset
6bit	5bit	5bit	16 bit.

If addresses of the program had to fit in this 16 bit field, it would mean that no program would be bigger than 2^{16}, which is far too small to be a realistic option today.
So, we can apply the following approach:

PC = register + Branch address.

This allows the program to be as large as 2^{32} and still be able to use conditional branches, which are found in ~~for~~ loops ~~or~~ or if statements and so they tend to branch to a nearby instruction.

Since the PC contains the address of the current instruction, we can branch within $\pm 2^{15}$ words of the current instruction if we use the PC as the register to be added to the address.

This form of branch addressing is called PC relative addressing, which is mostly used in recent computers as the destination of these instructions is likely to be close to the branch.

~~Set~~-F

(1) addi. \$sp, \$t2, 8.

addi is an I type instruction. So the format will be of I type.

Here, opcode will be 8. that means 001000

rt is \$sp ~~[illegible]~~ which is 29, so rt = 11101

rs is \$t2 which is 10, so rs = 01010

and offset is 8 so 0000,000 0,000 00,1000

so, code = 001000 01010 11101 0000 0000,0000 1000.

(2) Both ISA and Microarchitecture are a part of computer architecture.

ISA: The Instruction Set Architecture is implement on a processor. It is the interface between hw and sw
Modern ISA: 80x86/Pentium, PowerPC, DEC Alpha, MIPS
Allows diff. implementations of the same architecture.
Sometimes prevents adding new innovations.

Microarchitecture: On the other hand, it includes the parts of the processor and how these interconnect and interoperate to implement ISA.

Set - 4

1) $\underbrace{0000\ 0000\ 0011\ 1101}\ \underbrace{0000\ 1001\ 0000\ 0000}$

Load the code into register $s0 we need lui and ori.

First storing 16 higher bits in $s0.

lui $s0, 61

so, stored value in $s0 is 0000 0000 0011 1101 0000 0000, 0000 0000

Then we will store lower order bits,

ori $s0, $s0, 2304.

Thus, in $s0 we get,

$\boxed{0000\ 0000\ 0011\ 1101 \mid 0000\ 1001\ 0000\ 0000}$

2) ~~Ans~~

1. Code segment: It is the first part at the bottom address → 400000hex, and it is the part that stores program's instructions.

2. Data segment: It has two parts with:

a. Static data: Starts at 10000000hex, contains objects whose size is known to the compiler and whose lifetime - the interval during which a program can access them - is the program's entire execution. For example, in C, global variable are statically allocated since they can be refered anytime during a program's execution rd. , rs , rt

b. Dynamic data: It is above static data and is allocated by the program as it executes. In C, it is malloc Library routine. Since, compiler cannot predict how much memory a program will allocate, the OS expands the dynamic data area to meet demand. As the upward arrow in the figure indicates, malloc expands the dynamic area with the sbrk system call, which causes the OS to add more pages to the program's virtual address space above the dynamic data.

3. Stack segment: Resides at the top of the virtual address space, starting at 7fff ffff hex. The maximum size of the program's stack is not known in advance. As program pushes values on to the stack, the OS expands the stack segment down toward the data segment.

Set - 14

9 10 11

① slt . $t1 , $t2 , $t3 .

slt is a R-type instr.

so, opcode = 000 000 .

rs = 01010 as $t2 is 10

rt = 010011 as $t3 is 11

rd = 01001 as $t1 is 9 .

shamft = 00000

func = 101010 as slt = 42

So, ~~in~~ machin code:

000 000 01010 01011 01001 00000 101010.

② Difference between computer archi vs building archi.

1. Nature of Design: CA deals with the design of computer systems, including CPUs, memory hierarchy, instruction sets, and the organization of harware components. Whereas, BA deals with physical structures like building, houses, bridges etc. It concerns for not only the aesthetics but also the structural, functional and environmental aspects of the real physical spaces and buildings.

2. Object of Design: CA involves the design of CPUs, GPUs, memory modules, buses and other electronic components that make up a computer.
Whereas BA deals with interior and exterior structures of a building considering utilization, aestheticss, functionality safety etc.

3. Materials and Medium: CA deals with electronic components, IC chips, digital logics. More like it involves metal like silicone and conductive materials.
BA involves ~~smooth on~~ wide range of materials like concrete, steel, wood, ~~go~~ glass etc. It's more like craftsmanship.

4. Design Process: CA often involves simulations, mathematical modeling, and testing in a virtual environment. They use tools like simulators or CAD (Comp. Aided Design). software to create and evaluate design.

BA involves prototyping, construction and real world testing. Architects work with physical models and engage in on-site supervision during construction to ensure that the design is realized as intended.

5. Scale: CA is concerned with systems that can be incredibly small to Large but the scale is typically much smaller compared to building architecture.

But BA works on structure that range from small residential homes to massive skyscrappers and entire city planning, covering a wide range of scales.