1.

For a 4KB page, and a 32 bit address space, calculate the amount of memory needed to store a process's page tables. Assume each entry in the page table requires 10 bytes. Show all calculations.

Single paging is being assumed for all of the numericals.

4 kb page = 4096 = 2 ^ 12

Since the address space is 32,

address space = 2 ^ 32.

Lets divide address space by page

(address space / page) = No of entries

(2 ^32) / (2 ^ 12) = 2 ^ 20

= 1 million (1048576)

Therefore page table size = No of entries \* size of each entry

= 1048576 \* 10

= 10.48 mb

2.For a 4KB page, and a 64 bit address space, calculate the amount of memory needed to store a process's page tables. Assume each entry in the page table requires 10 bytes. Show all calculations.

Single paging is assumed for all the numericals.

4 kb page = 4096 = 2 ^ 12

Since address space is 64,

address space = 2 ^ 64.

Lets divide address space by page

(address space / page) = No of entries

(2 ^64) / (2 ^12) = 2 ^ 52

2 ^52 = (4.50 \* 10 ^15)

Therefore page table size = No of entries \* size of each entry

= (4.50 \* 10 ^ 15) \* 10 bytes

= 4.50 \* 10 ^ 16 bytes

3.For a 8KB page, and a 32 bit address space, calculate the amount of memory needed to store a process's page tables. Assume each entry in the page table requires 10 bytes. Show all calculations.

Single paging is assumed for all numericals.

8kb page = 8192 = 2 ^ 13

Since address space is 32,

address space = 2 ^ 32.

Lets divide address space by page

(address space / page) = No of entries

(2 \^\ 32) / (2 \^\ 13) = 2 \^\ 19

2 \^\ 19 = 524288

Therefore page table size = No of entries \* size of each entry

=524288 \* 10 bytes

= 5.24 mb

4.For a 8KB page, and a 64 bit address space, calculate the amount of memory needed to store a process's page tables. Assume each entry in the page table requires 10 bytes. Show all calculations.

Single paging is assumed for all numericals.

8kb page = 8192 = 2 ^ 13

Since address space is 64,

address space = 2 ^ 64.

Lets divide address space by page

(address space / page) = No of entries

(2 ^ 64) / (2 ^ 13) = 2 ^ 51

2 ^ 51 = 2.251 \* 10 ^ 15

Therefore page table size = No of entries \* size of each entry

= 2.251 \* 10 ^ 16 bytes

5. Describe the concept of pipe-lining, and why it is useful?

Let me explain the concept of pipelining with an example of doing laundry. Therefore I am also explaining the non-pipelined approach as well as the pipelined approach. The non-pipelined approach will be as follows:

1. Place one dirty load of clothes in the washer.

2. When you are done with the washer, place the wet load in the dryer.

3. When you are done with the dryer, place the dry load on the table and then fold them.

4. After folding is finished, put away the next dirty load.

5. Now start with the next dirty load.

Now let me explain to you the pipelined approach, which takes much less time.

1. As soon as the washer is finished with the first load and placed in the dryer, you load the washer with the second dirty load.

2. When the first load is dry, you place it on the table to start folding, move the wet load to the dryer, and put the next dirty load into the washer.

3. Next you have your roommate put the first load away, you start folding the second load, the dryer has the third load, and you put the fourth load into the washer.

This is the way in which an actual pipeline works. In pipeline the steps work concurrently. The reason that the pipelining is fast, the processes are running concurrently. If the number of loads is much larger than 4 then the stages will be full most of the time and the increase in throughput will be very close to 4

The same principles apply to processors where we pipeline instruction-execution.

MIPS instructions classically take five steps:

1. Fetch instruction from memory.

2. Read registers while decoding the instruction. The regular format of MIPS

Instructions allows reading and decoding to occur simultaneously.

3. Execute the operation or calculate an address.

4. Access an operand in data memory.

5. Write the result into a register.

IF---ID---EX---MEM---WR

The IF stands for instruction fetch.

The ID stands for instruction decode.

The EX stands for execution stage

The MEM stands for memory access stage

The WB stands for write back stage.

An architecture where pipeline is not supported will run the next instruction in 6th stage after completion of the fifth stage.

Pipelining is useful because it allows one to cut the delay of the critical path in a circuit path into two cycles, one instruction follows another in an assembly line fashion. Due to this the maximum circuit delay is split into multiple pieces and therefore a faster global clock rate is achieved. Throughput is increased due to pipelining. More pipeline stages means each stage can run faster, increasing the clock speed of the chip. Having more pipeline stages increases the number of instructions that can be executed per time.

Instruction level parallelism is enabled by the pipeline. From assembly instruction to actual execution, one can divide the process into well-defined stages. So, in a 5 stage pipeline, in one clock cycle we can effectively execute 5 different instructions. There can be stalls in each stage due to complexity of operation, division, and branching takes more time than addition or unavailability of data, or dependency between successive instructions.

Implementing a pipeline stage adds cost and delay to architecture, but overall CPI (clocks per instruction) decreases, which makes pipelining viable for computer architecture. One thing should be kept in mind while designing a pipeline is that each stage should be load balanced, i.e. the time required to process should remain almost equal.

6.Describe the IA-32e paging structure, in detail.

The IA32e paging uses a hierarchy of paging structures for producing translation of linear address. If you see the hierarchy, CR3 is in the bottom. With CR3 the first paging structure is done, the PML4 table. The IA32e paging translates 48bit linear addresses to 52-bit physical addresses. The 52 bits corresponds to 4 pbytes, the linear addresses are limited to 48 bits. At most 256 TBytes of linearaddress space may be accessed at a given time. A logical processor uses IA32e paging if CR0.PG = 1, CR4.PAE = 1, and IA32EFER.LME = 1.

The following bit positions are for the states when CR4.PCIDE = 0

2:0 :- The contents are ignored

3 (PWT) :- This is page table write-through; indirectly this determines memory type used to access the PML4 table at the time of the linear address translation.

4 (PCD) :- In this the page level cache is disabled; this indirectly determines the memory type which is used to access the PML4 table during the linear address translation.

11:5 :- this is ignored

M- 1:12 : The physical address of the 4kb aligned PML4 table is used for linear address translation.

63:M :- This is reserved.

11:0 :- PCID

M-1:12 :- Physical address of the 4kbyte aligned PML4 table used for linear address translation.

63:M Reserved

After software modifies the value of CR4.PCIDE, the logical processor immediately begins using CR3 as specified for the new value. Ex, if the software changes CR4.PCIDE from 1 to 0, the current PCID immediately changes from CR3[11:0] to 000H. The logical processor subsequently determines the memory type used to access the PML4 table using CR3.PWT and CR3.PCD, which had been bits 4:3 of the PCID.

The IA-32e paging might map linear address to 4-Kbyte pages, 2-Mbyte pages, or 1-GByte pages. With the following items the description of the IA-32e paging process.

A PML4 table which is 4kb is located at the physical address in bits 51:12 of CR3. PML4 table consists of 512 64-bit entries. The PML4E is selected based on the physical address which is defined as follows:

1. Bits 51:12 from CR3

2. Bits 11:3 are 47:39 of the linear address.

3. Bits 2:0 are all 0.

It is because a PML4E is identified using bits of 47:39 of the linear addresses, it controls access to a 512 GByte region of linear address space.

A naturally aligned page-directory pointer table is located at the physical address specified in the bits 51:12 of the PML4E. A page directory pointer table comprises 512 64 bit entries.. PDPTE is selected using the physical address defined as follows:

1. Bits 51:12 are from PML4E

2. Bits 11:3 are bits 38:30 of the linear address.

3. Bits 2:0 are all 0

The PDPTE is defined using the bits 47:30 of the linear addresses, the access to 1-Gbyte region of the linear address space is controlled by it.

If the PDPTE’s PS flag is 1, the PDPTE maps a 1-GByte page. In the following way the final physical address is computed as follows:

1. Bits 51:30 are from the PDPTE

2. Bits 29:0 are from the original linear addresses.

If the PDE’s PS flag is 0, a 4-KByte naturally aligned page directory is located at the physical address specified in bits 51:12 of the PDPTE . A page directory comprises 512 64-bit entries (PDEs). A PDE is selected using the physical address defined as follows:

1. Bits 51:12 are from the PDPTE

2. Bits 11:3 are bits 29:21 of the linear address.

3. Bits 2:0 are all 0.

As a PDE is mostly identified using bits 47:21 of the linear addresses, the access to a 2-Mbyte region of the linear address space is controlled by it.

If the PDPTE’s flag is 1, the PDE maps a 2-Mbyte page.

1. Bits 51:21 are from PDE.

2. Bits 20:0 are from original linear address.

If the PDE’s PS flag is 1, the PDE maps a 2-MByte page

1. Bits 51:21 are from the PDE

2. Bits 20:0 are from original linear addresses

If the PDE’s PS flag is 0, a 4-KByte naturally aligned page table is located at the physical address specified in bits 51:12 of the PDE . A page table comprises 512 64-bit entries (PTEs). A PTE is selected using the physical address defined as follows:

1. Bits 51:12 are from the PDE

2. Bits 11:3 are bits 20:12 of the linear address

3. Bits 2:0 are all 0

Because a PTE is identified using bits 47:12 of the linear address, every PTE maps a 4-KByte page . The final physical address is computed as follows:

1. Bits 51:12 are from the PTE.

2. Bits 11:0 are from the original linear address.

If a paging-structure entry’s P flag (bit 0) is 0 or if the entry sets any reserved bit, the entry is used neither to reference another paging-structure entry nor to map a page. A reference using a linear address whose translation would use such a paging-structure entry causes a page-fault exception.

The following bits are reserved with IA-32e paging

• If the P flag of a paging-structure entry is 1, bits 51:MAXPHYADDR are reserved.

• If the P flag of a PML4E is 1, the PS flag is reserved.

• If 1-GByte pages are not supported and the P flag of a PDPTE is 1, the PS flag is

reserved.1

• If the P flag and the PS flag of a PDPTE are both 1, bits 29:13 are reserved.

• If the P flag and the PS flag of a PDE are both 1, bits 20:13 are reserved.

• If IA32EFER.NXE = 0 and the P flag of a paging-structure entry is 1, the XD flag

(bit 63) is reserved.

A reference using a linear address that is successfully translated to a physical

address is performed only if allowed by the access rights of the translation; see

Section 4.6.

Figure 4-11 gives a summary of the formats of CR3 and the IA-32e paging-structure

entries. For the paging structure entries, it identifies separately the format of entries

that map pages, those that reference other paging structures, and those that do

neither because they are “not present”; bit 0 (P) and bit 7 (PS) are highlighted

because they determine how a paging-structure entry is used.

In short:

Every entry in CR3 points to the location of PML4E. Therefore, CR3 is essential to get PML4E

Every entry in the PML4E points to the location of a PDPTE. Therefore PML4E is essential to get PDPTE

Every entry in a PDPTE(Page directory pointer table) points to the location of a PDE (or a 1G page0). Therefore PDPTE is required for PDE

Every entry in a PDE points to the location of a PT (or a 2M page).

Every entry in a PT points to a 4k page.

To map 4k of the address space, you need one PT, one PD to refer to that PT, one PDPT and one PML4, which requires 16k of space, if you want to map everything using 4k pages, each table refers to 512 sub tables: (1 + 512\*(1 + 512\*(1 + 512 \* (1)) \* 4096

= 0x0000 0080 4020 1000 bytes, just a little over 513GB.

7.What are some examples of pipeline hazards, and what are some ways of dealing with them?

Pipeline Hazards are situations in pipelining when the next instruction cannot execute in the following clock cycle. These events are known as hazards and there are three different types.

Structural hazard: This hazard occurs when the hardware cannot support the combination of instructions that we want to execute in the same clock cycle. Ex, a structural hazard in the laundry room would occur if we used a washer-dryer combination instead of a separate washer and a dryer, or if our roommate was busy doing something else and wouldn’t put clothes away.

Solution: A MIPS instruction set was designed to be pipelined so that it is fairly easy for the designers to avoid structural hazards while designing a pipeline. Suppose, however that we had a single memory instead of two memories. If pipeline in a basic pipeline with three instructions had a fourth instruction, we would see that in the same clock cycle the first instruction is accessing data from memory while the fourth instruction is fetching an instruction from the same memory.

Data Hazard: This occurs when the pipeline must be stalled because one step must wait for another to complete. In a computer pipeline, data hazards arise from the dependence of one instruction on an earlier one that is still in the pipeline.

Lets take an example,

add s0, t0, t1

sub t2, s0, t3

Solution: This data hazard has to be solved by the method known as forwarding or bypassing. This is a method of resolving a data hazard by retrieving the missing data element from internel buffers rather than waiting for it to arrive from programmer visible registers or memory. Adding extra hardware to retrieve missing item early from the internal resources is called forwarding or bypassing.

Forwarding with two instructions:

add s0, t0, t1 IF---ID---EX---MEM---WR

sub t2, s0, t3

This is a RAW hazard basically.

Here the forwarding is done from the output of the execution stage of the add instruction to the input of the execution stage.

Example 2: (load instruction)

Lw s0, 20(t1)

Sub t2, s0, t3

This is again a RAW hazard.

Since it is a load instruction , the forwarding is done from the MEM to the input of the EX stage. This is because in the load instruction , the result is determined In the MEM stage. Therefore a bubble has also to be added in the instruction. The bubble is also called as pipeline stall. This hazard is also called as load use data hazard.

RAW, WAR,RAR and WAW are the four type of hazards in this section

RAW: Read after write (a true dependency)

WAR: Write after read (an anti-dependency)

WAW: Write after write (an output dependency)

• Therefore to resolve data hazards: insert a pipeline bubble whenever a read after write (RAW) dependency is encountered, guaranteed to increase latency, or

• utilize out of order execution to potentially prevent the need for pipeline bubbles

• utilize operand forwarding to use data from later stages in the pipeline

In the case of, out of order execution the algorithm used can be:

• scoreboarding, in which case a pipeline bubble will only be needed when there is no functional unit available

• the Tomasulo algorithm, which utilizes register renaming allowing the continual issuing of instructions

Control hazards:

The third type of hazard is called a control hazard, arising from the need to make a decision based on the results of one instruction while others are executing.

The following are the solutions to control hazards in the laundry room and its computer equivalent.

Stall: Just operate sequentially until the first batch is dry and then repeat until you have the right formula. This is to go let the instruction execute, but this is usually slow.

The equivalent solution will be to use a branch instruction. There are different circumstances when a branch is taken and also when a branch is not taken.

Ex1.

Add 4, 5, 6

beq 1, 2, 40

or 7, 8, 9

Sol: In this problem due to the branch which is the second instruction, the bubble is created instead of the third instruction. The fourth instruction will come after a delay of 400 ps.

The cost of the branches is usually high, therefore predicting is also an effective way. But there is penalty involved in case of incorrect prediction.

Now in this example, it is predicted that the branch is not taken.

add 4, 5, 6

beq 1, 2, 40

lw 3, 300(0)

Therefore, this pipeline goes on without any stalls.

Now take another example,

add 4, 5, 6

beq 1, 2, 40

Or 7, 8, 9

Sol: In the solution of this , a bubble is produced after the second instruction (after branch). Therefore the third instruction is created after 400ps.

Therefore, the two approaches are that the branches will be taken or not taken. There is also a third approach which is delayed decision.