ELEC 3441 Hw_3 Part A

JIANG Feiyu

A.1 Page Table Size

A 1.1

According to the question, the maximum amount of memory should be:

Segment	Max Address Diff	Size
code	`0x101FFFFF` - `0x10000000` + `0x1` = `0x200000`	`2097152` = 2^{21} bits = 2^{18} bytes
stack	`0xA00000000000` - `0xA0000000FFFF` + `0x1` = `0x10000`	`65536` = 2^{16} bits=2^{13} bytes
heap	`0xD00000001FFFFFFF` - `0xD000000000000000` + `0x1` = `20000000`	`536870912` = 2^{29} bits= 2^{26} bytes

A process may use $2^{18} + 2^{13} + 2^{26}$ Bytes Memory.

A 1.2

Suppose the size of a page is 2^p, then
The maximum number of pages may be allocated by each segment is:

- Code segment: 2^{21-p}
- Stack segment: 2^{16-p}
- Heap segment: 2^{29-p}

$2^{21-p} + 2^{16-p} + 2^{29-p}$ pages may be allocated in total. When p is 0, it reaches maximum.

A 1.3

Since this system has `33-bit` physical address, and the size of a page is $2^p = 2^{12}$, which

means the number of Physical Page is $\frac{2^{33}}{2^{12}} = 2^{21}$.

Thus, $sizeof(PPN) = 21$, $sizeof(PTE) = \lceil \frac{szieof(PPN)+3}{8} \rceil = 3$ bytes.

$size\ of\ linear\ page = 3\ byte \times 2^{64-12} = 3 \times 2^{52}\ byte$

A 1.4

According to the question, the first level of page table contains 8192 (2^{13}) entries.

Since the page size is 2^p, the number of entites is 2^{64-p}, there are $\frac{2^{64-p}}{2^{13}} = 2^{51-p}$ PTEs in each second level page table.

A 1.5

Using the 2-level page table mentioned in `1.4`:

The size of each page is 2^p, $PPN = 33 - p$, and the PTE is $sizeof(PTE) = \lceil \frac{36-p}{8} \rceil$

In total, the maximum storage needed is

$2^{13} \times \lceil \frac{36-p}{8} \rceil + 2 \times 2^{51-p} \times \lceil \frac{36-p}{8} \rceil = \lceil \frac{36-p}{8} \rceil \times (2^{13} + 2^{52-p})$.

When `p=0`, it reaches the maximum storage.

A 1.6

Assume instead that a process only uses the lower 25 % of its address space in all 3 segments. According to the result mentioned in `A 1.1`:

A process may use $2^{18} + 2^{13} + 2^{26} = 67379200$ Bytes Memory, whcih means it will be $\frac{25\% \times 67379200}{1024 \times 1024} = 16$MB ($2^{24}$ bytes=2^{27} bits).

Then there are 2^{27-p} pages need in total, while first level is 8192(2^{13}) entries, the second level contains 2^{14-p} entries.

So the one of the second level page $size = 2^{14-p} \times \lceil \frac{36-p}{8} \rceil bytes$

So the total memory needed is:

$2^{13} \times \lceil \frac{36-p}{8} \rceil + 2 \times 2^{14-p} \times \lceil \frac{36-p}{8} \rceil$

A 1.7

Assume $p = 12$, $f(p) = 3 \times (2^{13} + 2^3) = 24600$

A 1.8

Segment	Address
code	0x0000 0000 1000 0000 - 0x0000 0000 101F FFFF
stack	0x0000 A000 0000 0000 - 0x0000 A000 0000 FFFF
heap	0x0010 0000 0000 0000 - 0xD000 0000 1FFF FFFF

In this case, all segment are in the same first level PTE, so that 1 first level page table and 1 second level page table are needed to storage when the process start. So the total $size\ of\ page\ table = \frac{(36-p)(2^{13}+2^{51-p})}{8} bytes$

So when $p = 28$ the total size of page table is minimum, so the $size = \frac{2^{13}+2^{23}}{8}$ = 1049600 bytes

A.2 TLB Trace

A 2.1

According to the question, there are:

- 4 KiB page size
- 4-entry, fully associative TLB, true LRU

$$TLB\ reach = 4 \times 4KiB = 16KiB$$

A 2.2

According to the page table, the first four bit is the VPN and the TLB uses the true LRU. Thus,

Memory access	VPN	Hit/Miss
0x2468	0x2	Hit
0x246C	0x2	Hit
0x3070	0x3	Miss
0xA500	0xA	Miss
0x2470	0x2	Hit
0x5CCC	0x5	Miss
0xC000	0xC	Page Fault
0x8004	0x8	Miss
0x5CD0	0x5	Hit

Final state of the TLB

Idx	VPN	PPN
3	0x5	0x22E
2	0x2	0xB07
1	0xC	0xD12(disk 2 + 0xD10)
0	0x8	0x992

Final state of the Page Table

VPN	PPN or Disk	VPN	PPN or Disk
0xF	0x28A	0x6	empty
0xE	empty	0x5	0x22E

VPN	PPN or Disk	VPN	PPN or Disk
0xD	empty	0x4	empty
0xC	0xD12	0x3	0xA9C
0xB	empty	0x2	0xB07
0xA	0x289	0x1	empty
0x9	empty	0x0	empty
0x8	0x992		
0x7	empty		

A 2.3

- **Advantage**:
    i. **TLB**: Increasing page size will decrease the width of VPN. For a TLB, that means the comparison tag is smaller. A larger page size improves the TLB hit rate.
    ii. **Reduce external fragmentation**: A larger page size reduces the number of page table entries in the page table, thus reducing the size of the page table and saving memory space and access time.
    iii. **Reduced I/O expense**: Larger pages allow for fewer I/O times, allowing more data to be loaded in the cache at once
- **Disadvantage**:
    i. **TLB**: When the page size increases, each page can accommodate more data. This means that the same number of pages can cover a larger virtual address range. In a direct-mapped TLB, each virtual address can only be mapped to a specific entry in the TLB. When the page size increases, more virtual addresses are mapped to the same TLB entry index. This increases the likelihood of multiple virtual addresses mapping to the same TLB entry. Therefore, when the page size increases, a direct-mapped TLB is more prone to experiencing conflict misses.
    ii. **Increase internal fragmentation**: Larger pages can lead to waste of

memory because each process has to allocate the entire page, even if they only occupy a small part of it.

iii. **Address space waste**: The virtual address space of each process must be aligned with the page size; if the page size is too large, then unused address space is wasted.

A 2.4

Since using a larger TLB, the capacity miss is reduced, especially for those process that using the continuous memory space repeatedly. However, as we discussed in `A 2.3`, larger TLB may also result in conflict misses. For instance, a process that uses very few pages, but every entry maps to the same index in the TLB, will always have conflict miss.

When using an 8-entry direct-mapped TLB compared to the original design, there are the following tradeoffs to consider:

- **Hit Rate**: The direct-mapped TLB with 8 entries may have a higher hit rate compared to the 4-entry fully associative TLB, assuming the workload does not have conflicts on those direct-mapped slots.
- **Conflict Misses**: If virtual pages in the workload are mapped to the same TLB entry in the direct-mapped scheme, an 8-entry direct-mapped TLB may be more prone to conflict misses compared to a 4-entry fully associative TLB.
- **Cost**: Direct-mapped caches are typically simpler and less costly than fully associative caches.
- **Performance**: Due to the simpler hardware required for direct mapping, an 8-entry direct-mapped TLB may have lower access time.

Let's consider some concrete senarios:

- **8-entries Direct Map TLB Would works better**:
  A working set consists of 8 distinct pages that are accessed in a **cyclic** manner. Let's assume these pages have virtual page numbers (VPNs) and indices of `0x1_000`, `0x2_001`, `0x3_010`, `0x4_011`, `0x5_100`, `0x6_101`, `0x7_110`, and `0x8_111`. If each VPN is mapped to a unique entry in an 8-entry direct-mapped TLB through different index bits (determined by the lower bits of the VPN), no conflicts occur, and each page is directly mapped into the TLB without any replacements.

However, if the same workload is applied to a 4-entry fully associative TLB, frequent replacements occur due to conflicts. This results in more misses and potential performance degradation.

- **4-entries Fully Associative TLB Would works better**:
  Consideraing an application with a **small working set** consisting of four pages with VPNs `0x1`, `0x2`, `0x3`, and `0x4`. The workload exhibits a repetitive access pattern where these pages are **accessed frequently**. When using a 4-entry fully associative TLB, all of these pages can remain in the TLB, regardless of the access pattern, as any page can be placed in any TLB slot.
  However, if we opt for an 8-entry direct-mapped TLB and if two or more of these pages happen to map to the same TLB slot (due to their index bits), there will be a constant need to replace TLB entries. This frequent replacement results in a higher miss rate and leads to poorer performance, despite the larger number of TLB entries available.

A.3 TLB and Cache

A 3.1

According to the question, page size is 4 KiB (2^{12}B)

1. $32 - 12 = 20$
2. $35 - 12 = 23$
3. 12

A 3.2

Assume it's 4-entry, fully associative TLB (since the question didn't provide the entry info), then the $TLB\ reach = 4 \times 4KiB = 16KiB$

A 3.3

If a TLB miss, the processor must reload the entry form the page table in memory, which equivalently means that a cache miss happens, which is 400 ns.

If a page fault happens, the time to update the page table is 10000ns.

A 3.4

According to the question, Page fault penalty is 10000 ns while on TLB hit, TLB access takes 0.5 ns.

Thus, when 3 % results in TLB MISS. Among the misses, 1 % results in a page fault.

- the rate for TLB hit is 97%
- the rate for TLB miss with no page fault is 2.99%
- the rate for TLB miss with page fault is 0.01%

$avg = 0.97 \times 0.5 + 0.0297 \times 400 + 0.0003 \times 10000 = 1.5365 \times 10^{-8}$s

Since we assume the data cache always hit, the average memory access time is

$avg_access = 1.5365 \times 10^{-8} + 0.5 \times 10^{-9}$s

A 3.5

Based on `3.4`, we assume the $avg = 15.365 \times 10^{-8}$s

According to the formula, $AMAT = Hit\ time + Hit\ Time + Miss\ Rate \times Miss\ Penalty$

According to the question, the rate for data cache hit is 95% while taking 0.5ns and the rate for data cache miss is 5% while taking 400ns.

Therefore,

$$Data_{AMAT} = 0.5 + 0.5 \times 400 = 20.5ns$$

The AMAT for both address translation and data cache is:

$$AMAT = 20.5ns + 15.365ns = 3.5865 \times 10^{-8}\text{s}$$

A 3.6

i. I think this question depends. A paged virtual memory system cannot function **properly** without a TLB. On the other hand, the size of TLB may cause a negative impact on AMAT.

ii. **The performance expected to be worse than the original scheme with TLB**. Without a TLB, each address translation would require direct access to the page table, leading to a significant performance decrease. Even though most page table entries may be cached in the data cache, there would still be additional time required to access the page table, without the fast address translation provided by the TLB. Therefore, the performance without a TLB is expected to be worse than the original scheme with a TLB.

iii. The size of the TLB affects the average memory access time (AMAT) by reducing the TLB miss rate. A larger TLB means the processor is more likely to find the required page table entry without accessing slower main memory, resulting in a lower overall AMAT. However, increasing TLB size also increases hardware complexity, which may impact performance.

A.4 Dense-Sparse Vector Multiplications

A 4.1

```
    1: add ×9, x0, x0 # 1 cycle
    2: add x8, x0, x0 # 1 cycle
1oop: 1w x3, 0(x1)    # 1 cycle
    4: 1w x4, 0 (x2) # 1 cycle
    5: mult x5, x3, x4 # x cycle
    6: add x9, x9, ×5 # 1 cycle
    7: addi x1, x1, 4 # 1 cycle
    8: addi x2, x2, 4 # 1 cycle
    9: addi x8, x8, 1 # 1 cycle
    10: bne x8, x10, 100p # 1 cycle, prediction always correct
```

$No.\ of\ cycles = 2 + N \times (7 + x)$

A 4.2

A correctly predicted branch takes 1 cycle. A mispredicted branch takes m cycles to complete.

```
    1: add x9, x0, x0 # 1 cycle
    2: add x0, x0, x0 # 1 cycle
loop: 1w ×3, 0(x1) # 1 cycle
    4: lw x4, 0(x2) # 1 cycle
    5: beq x4, x0, skip # 1 cycle
    6: mult x5, x3, x4 # x cycle
    7: add x9, x9, x5 # 1 cycle
skip: addi x1, x1, 4 # 1 cycle
    9: addi x2, x2, 4 # 1 cycle
    10: addi x8, x8, 1 # 1 cycle
    11: bne x8, x10, 100p # 1 cycle
```

- Considering the `loop` function
    - if the prediction is correct: $4 + x$
    - if the prediction is wrong: $2 + m$

Suppose the sparsity ratio is r for vector B, then
$cycles_B = r(2+m) + (1-r)(4+x) = 4 + x + (m - x - 2)r$

- Considering the `skip` function: 4 cycles

Thus, $No.\ of\ cycles = 2 + N \times [4 + x + (m - x - 2)r] + 4N = 2 + N \times [8 + x + (m - x - 2)r]$

A 4.3

If x = 50, m = 10, then

$No.\ of\ cycles_A = 57N + 2$

$No.\ of\ cycles_B = 2 + N \times (58 - 42r)$

When $\frac{No.\ of\ cycles_A}{No.\ of\ cycles_B} = \frac{57N+2}{2+N\times(58-42r)} > 1$, cycles for program B is less than cycles in program B,

which means when $r > \frac{1}{42}$, program B will be faster.

A 4.4

According to the Iron Law: $Time = Instructions \times CPI \times Cycle\ Time$, assume that cycle time here is the same for all of the three programs.

Program	Instruction Count	CPI	Cache Behavior
A	simple loop, constant No. of instructions per iteration,unaffected by the sparsity level	No prediction concern, `mult` instruction take multiple cycles	a regular memory access pattern for vecA and vecB
B	simple loop, constant No. of instructions per iteration, includes a	misprediction could occur, long multiplications could be skipped when `r` is high to reduce CPI	branching could lead to less efficient

Program	Instruction Count	CPI	Cache Behavior
	conditional branch (skip `0`)		use of the instruction pipeline
C	processes only non-zero elements, instruction count could be significantly reduced with high sparsity	`slli` and `add` may take longer than the simple `lw`, multiplications are only performed for non-zero elements, reducing CPI when r is high.	maybe more cache misses due to the irregular access pattern

Performance Comparison:

- **Low Sparsity (r close to 0%)**: Programs A and B might perform better because the condition checking overhead of programs A and B is small / no branching
- **High Sparsity (r close to 100%)**: Program C may perform the best because it doesn't waste cycles on zero elements, though cache behavior might be worse if `vecB_idx` doesn't follow a pattern that the cache can predict.
- **Very Small N (close to 10)**: The setup overhead in Program C might make it less efficient than A or B.
- **Very Large N (close to 10 million)**: The efficiency of Program C's skipping over zeros could outweigh cache inefficiencies.