15-740/18-740 Computer Architecture, Fall 2011 Midterm Exam II

Instructor: Onur Mutlu Teaching Assistants: Justin Meza, Yoongu Kim Date: December 2, 2011



Instructions:

- 1. This is a closed book exam. You are allowed to have one letter-sized cheat sheet.
- 2. No electronic devices may be used.
- 3. This exam lasts 1 hour 50 minutes.
- 4. Clearly indicate your final answer.
- 5. Please show your work when needed.
- 6. Please write your initials on every odd page.
- 7. Please make sure that your answers to all questions (and all supporting work that is required) are contained in the space provided.
- Be cognizant of time. Do not spend too much time on one question.
- Be concise. You will be penalized for verbosity.
- Show work when needed. You will receive partial credit.
- Write legibly. Show your final answer.

I. Potpourri (69 points)

1. Dual-Core Execution vs. Runahead Execution

a) What three advantages does *Dual-Core Execution* provide that *Runahead Execution* cannot achieve?

Continuous prefetching: Can continue to prefetch after the L2 miss comes back.

Separate L1 caches in the two cores can reduce potential pollution from prefetching.

None.

b) What are three disadvantages of *Dual-Core Execution* that are not present in *Runahead Execution*? Requires two cores.

Longer branch misprediction recovery pipeline.

Executes instructions in two cores concurrently \rightarrow higher maximum power consumption.

2. Replicated Register Files

Alpha 21264 employed a technique called *clustering*. The machine had two copies of the integer register file. Some instructions accessed their operands from one copy, others accessed their operands from the other copy. The data in the two register file copies were kept coherent: when an instruction wrote to its destination register, the destination register was updated in both copies of the register file, albeit the update was delayed by one cycle in one of the register files.

a) Explain what benefit employing two register files provided, if any.

Ability to access many (8) operands in parallel without making the register file too slow or too costly (i.e., truly multi-ported).

b) Is there a cheaper way to provide the same benefit? How so? Explain in no more than 30 words.

Yes. Multiple banks in the register file.

c) What are the downsides of the "cheaper way" you just described?

1. Bank-conflicts still serialize register file accesses.

2. Requires bank conflict detection logic and crossbar between inputs to banks.

3. Memory Channels and Banks

A computer architect is deciding how to design the main memory system of her company's next-generation single-core processor. She is considering two choices. The first is to have two memory channels, where each channel has a single bank. The second is to have a single memory channel, with a DRAM chip that has two banks. Assume the size and structure of a bank and the width of a channel are the same in both designs.

a) What is the benefit of the first design over the second?

Two accesses to memory can be performed truly independently.

b) What is the disadvantage of the first design over the second?

More expensive: increased pin-count to implement additional channel.

Assume the designer picks the first design. She evaluates two different options for mapping a physical address to different channels. One option she considers (Option A) is to place consecutive 64-byte cache blocks in the physical address space in consecutive channels. A second option she considers (Option B) is to place consecutive 4KB pages in the physical address space in consecutive channels. She simulates two single-threaded programs to determine which option provides higher performance. Assume there is no prefetching and all accesses to main memory are initiated by load instructions.

c) For one program, she finds Option A provides significantly higher performance than Option B. Explain why this could be the case. What can you say about the access pattern of the program?

The program generates multiple accesses in parallel to the two banks. For example, streaming (array-traversal). For Option A, a streaming application fully utilizes the additional channel's bandwidth, as well as experiencing row-buffer hits in both banks.

d) For the other program, she finds Option B provides significantly higher performance than Option A. Explain why this could be the case. What can you say about the access pattern of the program?

The benefit can come from either improved bank parallelism or improved row locality. For example (of both), two concurrent streams, e.g., while (i++ < N) a[i] = 2*b[i]; For Option B, the two different arrays can be mapped to different banks and experience row-buffer hits in their respective banks. On the other hand, for Option A, the two streams will thrash the row-buffer of both banks.

4. Memory Channel Partitioning

Assume we have a multi-core system that has multiple memory channels. Each core in the system executes one single-threaded application. In class, we discussed having the system software map the data of applications that have low memory intensity to separate memory channels than that of applications that have high memory intensity. This enables applications with low memory intensity to access different channels than those with high memory intensity.

a) Why is this a good idea? Provide all possible reasons.

- 1. Eliminates memory interference caused by intensive applications to non-intensive applications
- 2. Preserves row-buffer locality of applications that have high row-buffer locality
- 3. Can enable keeping a free-for-all memory controller \rightarrow simpler hardware
- 4. Implemented in system software: flexibility

b) Why is this a bad idea? Provide all possible reasons.

- 1. Bandwidth fragmentation
- 2. Reduced bank-level parallelism
- 3. Reduced effectiveness when a partition's capacity overflows
- 4. Which partition to allocate memory when an application is still being profiled?
- 5. When application mix changes, partitions may have to be shuffled around (lots of migration)
- 6. An application may undergo a phase change that may cause it to be in the wrong partition
- 7. Increases intra-application interference

c) What does application-aware memory channel partitioning achieve that application-aware memory request scheduling cannot provide?

Eliminates inter-application memory interference between some (e.g., intensive and non-intensive) applications. Memory scheduling only mitigates the memory interference.

5. Virtual Memory

a) What is cached in the TLB (Translation Lookaside Buffer)?

Page table entries.

b) Why is it undesirable to have a physically-indexed, physically-tagged L1 cache? Explain.

Virtual-to-physical translation (TLB lookup) is on the critical path.

c) Why is it undesirable to have a physically-indexed, physically-tagged L2 cache? Explain.

It is not. Translation has already been done when L2 cache is accessed.

d) What is the benefit of having a virtually-indexed, physically-tagged L1 cache? Explain.

Virtual-to-physical translation (TLB lookup) can be parallelized with cache tag/data lookup.

e) What is the benefit of having a virtually-indexed, physically-tagged L2 cache? Explain.

There is none. Translation has already been done when L2 cache is accessed.

f) What is the benefit of having a virtually-indexed, virtually-tagged L1 cache? Explain.

Address translation not needed to access the cache \rightarrow translation completely off the critical path.

g) What is the benefit of having a virtually-indexed, virtually-tagged L2 cache? Explain.

There is none. Translation has already been done when L2 cache is accessed.

6. Branch Elimination

a) What are two compiler-based techniques that can eliminate some branches?

Predicated execution

Predicate combining

b) What is a common disadvantage of both techniques?

1. Additional code executed even if not needed. 2. More data dependencies in code.

c) What type of conditional branches can neither technique eliminate?

Backward (loop) branches

d) Why not?

Eliminating the branch would eliminate the loop.

7. Feedback-Directed Prefetching

In lecture, we covered the idea of feedback-directed prefetching. As you might remember, you also reviewed the paper published by Srinath et al. in HPCA 2007, "Feedback Directed Prefetching: Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers."

a) What problem does feedback-directed prefetching aim to solve?

Performance degradation caused by uncontrolled aggressive prefetching (due to contention in cache and memory).

b) Explain the basic idea of feedback-directed prefetching in no more than 30 words.

Throttle up/down the aggressiveness of the prefetcher by taking into account the timeliness and accuracy of the prefetcher.

c) In lecture, we also discussed why the problem becomes more difficult in a multi-core system. Explain why.

The decision to throttle new needs to take into account the interference of the prefetcher with other cores' requests in the shared memory system (caches, interconnect, main memory). This requires additional information gathering and decision making based on that.

d) What additional information needs to be taken into account to make feedback-directed prefetching effective in multi-core systems? Be specific and explain.

1. How much prefetches of a core interfere with other cores' requests in the cache

2. How much prefetches of a core interfere with other cores' requests in the main memory (row buffer and bank)

3. How much the other cores need memory bandwidth (and maybe cache space)

II. Data Marshaling (16 points)

In class, we discussed a technique called *Data Marshaling*, which reduces the performance impact of intersegment data transfers in staged execution models, such as Accelerated Critical Sections or Producer Consumer Pipeline Parallelism. Recall that Data Marshaling is a technique that proactively ships data produced in one core to the cache of the core executing the next segment.

Yoongu and Justin are evaluating the effect of Data Marshaling on three different baseline systems. All systems implement Accelerated Critical Sections. When Data Marshaling is implemented, it places data into the L2 cache slice of the next segment's core. Only L2 cache size is different among the three systems.

Here is the performance they find on a large number of workloads.

	L2 cache slice size	DM's performance improvement over baseline
System 1	256 KB	9%
System 2	512 KB	12%
System 3	1024 KB	14%

Explain why DM's performance benefit increases as L2 cache slice size increases. Note that Yoongu and Justin already figured out that DM does not pollute caches in any of the systems.

Larger cache reduces all other misses but communication misses. Communication misses become more of a bottleneck, and thus DM, which reduces them, leads to higher performance.

III. Sectored Caches (25 points)

A processor with a 24-bit byte-addressable memory ships with a 2-way set-associative *sectored cache*. The cache uses the perfect LRU replacement policy with writeback. The number of sectors per block is a power of 2.

If the tag-store size is 26112 bits (including meta-data), determine the cache block size. The tag store does not have redundant information.

 $26112 = 2^9 \times 3 \times 17$

Let T be the number of tag bits required for a single cache block. Let S be the number of sectors per cache block (power of two). Let tag_{set} be the number of tag bits required for a single cache-set. $tag_{set} = WAYS \times T + WAYS \times S \times (VALID_{BITS} + DIRTY_{BITS}) + LRU_{BITS}$ $tag_{set} = 2 \cdot T + 2 \cdot S \cdot (1+1) + 1$ $tag_{set} = 2 \cdot T + 4 \cdot S + 1$ Let N be the number of sets in the entire cache. Let tag_{total} be the number of tag bits required for the entire cache. $tag_{total} = 2^N \cdot tag_{set}$ $tag_{total} = 2^N \cdot (2 \cdot T + 4 \cdot S + 1)$ $tag_{total} = 2^N \cdot (2 \cdot T + 4 \cdot S + 1) = 2^9 \times 3 \times 17$ $\therefore N = 9$ $\therefore 2 \cdot T + 4 \cdot S + 1 = 51, \quad T + 2S = 25$ Possible solutions for T + 2S = 25 (where S is a power of two): $(T, S) = (23, 1) \rightarrow$ Impossible since T + N > 24 $(T, S) = (21, 2) \rightarrow \text{Impossible since } T + N > 24$ $(T, S) = (17, 4) \rightarrow$ Impossible since T + N > 24 $(T, S) = (9, 8) \rightarrow Only possible solution$ $\therefore T = 9$ $\therefore S = 8$ Let B be the number of bytes in a cache block. T + N + B = 249 + 9 + B = 24 $\therefore B = 6$ \therefore Cache block size = 64-bytes

IV. Prefetching (50 points)

A processor is observed to have the following access pattern to cache blocks. Note that the addresses are **cache block addresses**, not byte addresses.

 $\begin{array}{l} A, A+4, A+8, A+12, A+16, A+20, A+24, A+28, \\ A, A+4, A+8, A+12, A+16, A+20, A+24, A+28, \\ A, A+4, A+8, A+12, A+16, A+20, A+24, A+28 \end{array}$

Assume we have the following hardware prefetchers. None of them employ confidence bits, but they all start out with empty tables at the beginning of the access stream shown above. Assume that 1) each access is separated long enough in time such that all prefetches issued can complete before the next access happens, 2) the prefetchers have large enough resources to detect and store access patterns, 3) the prefetchers prefetch into a fully-associative cache whose size is 4 cache blocks.

Calculate the prefetcher accuracy and coverage of each prefetcher for the access stream above. Show your work.

a) A stream prefetcher with prefetch degree of 1 (i.e., a next-block prefetcher).

Circle which of the cache block addresses are prefetched by the prefetcher:

```
A, A+4,
             A + 8,
                     A + 12,
                               A + 16,
                                         A + 20,
                                                   A + 24,
                                                            A + 28,
A, \quad A+4,
            A + 8,
                     A + 12,
                               A + 16,
                                         A + 20,
                                                   A + 24,
                                                            A + 28,
    A + 4.
            A + 8.
                     A + 12,
                               A + 16,
                                         A + 20,
                                                   A + 24,
                                                            A + 28
A.
```

ACCURACY: 0/(8+8+8) = 0COVERAGE: 0/(8+8+8) = 0

b) A next line prefetcher with prefetch degree of 2 (i.e., a next-2-blocks prefetcher).

Circle which of the cache block addresses are prefetched by the prefetcher:

A, A+4,A + 8,A + 12,A + 16,A + 20,A + 24,A + 28,A, A+4, A+8,A + 12,A + 16,A + 20,A + 24,A + 28, $A, \quad A+4,$ A + 8,A + 12,A + 16,A + 20,A + 24,A + 28

ACCURACY: 0/(16+16+16) = 0

COVERAGE: 0/(8+8+8) = 0

c) A next line prefetcher with prefetch degree of 4 (i.e., a next-4-blocks prefetcher).

Circle which of the cache block addresses are prefetched by the prefetcher:

A,	A+4,	A+8,	A+12,	A+16,	A+20,	A+24,	A+28
A,	A+4,	A+8,	A+12,	A+16,	A+20,	A+24,	A+28
A,	A+4,	A+8,	A+12,	A+16,	A+20,	A+24,	A+28

ACCURACY:
$$7/(4 \times 8) = 7/32$$

COVERAGE: $7/8$

d) A stride prefetcher (that works on cache block addresses).

Circle which of the cache block addresses are prefetched by the prefetcher:

A,	A+4,	A+8,	A+12,	A+16,	A+20,	A+24,	A+28,
A,	A+4,	A+8,	A+12,	A+16,	A+20,	A+24,	A+28,
A,	A+4,	A+8,	A+12,	A+16,	A+20,	A+24,	A+28

ACCURACY: $(6 \times 3)/(7+8+8) = 18/23$

COVERAGE: 6/8 = 3/4

e) A Markov prefetcher with a correlation table of 2048 entries (assume each entry can store one next address).

Circle which of the cache block addresses are prefetched by the prefetcher:

A,	A+4,	A+8,	A + 12,	A+16, A	A + 20, A	+24, A	+28,
A,	A+4,	A+8,	A+12,	A+16,	A+20,	A+24,	A+28,
Α,	A+4,	A+8,	A+12	, A+16,	A+20,	A+24	, A+28

ACCURACY: (7+8)/(8+8) = 15/16

COVERAGE: $(7+8)/(8\times3) = 15/24 = 5/8$

f) A Markov prefetcher with a correlation table of 4 entries (assume each entry can store one next address).

Circle which of the cache block addresses are prefetched by the prefetcher:

A,A + 4,A + 8,A + 12,A + 16,A + 20,A + 24, A + 28, A + 12,A + 16,A + 20,Α. A + 4,A + 8,A + 24,A + 28,A + 4,A+8,A + 12,A + 16,A + 20,A + 24,A + 28A,

ACCURACY: 0/0 (undefined)

COVERAGE: 0/(8+8+8) = 0

g) What is the minimum number of correlation table entries that should be present for the Markov prefetcher to attain the same accuracy and coverage as the Markov prefetcher with infinite number of correlation table entries? Show your work to get partial credit.

8

h) Based on the above access stream, which prefetcher would you choose if the metric you are optimizing for was memory bandwidth? Why?

Markov. Since it has the highest accuracy, it utilizes memory bandwidth the best.

i) Which prefetcher would you choose if you wanted to minimize hardware cost while attaining reasonable performance benefit from prefetching. Explain your reasoning in no more than 40 words.

Stride. Has highest coverage (potentially highest benefit from prefetching) at reasonably high accuracy (likely not too much harm), while adding a small amount of logic.

V. Banks (38 points)

A processor's memory hierarchy consists of a small SRAM L1-cache and a large DRAM main memory, both of which are banked. The processor has a 24-bit physical address space and does not support virtual memory (i.e., all addresses are physical addresses). An application has just started running on this processor. The following figure shows the timeline of memory references made by that application and how they are served in the L1-cache or main memory.



For example, the first memory reference made by the application is to byte-address **0xffbc67** (assume that all references are byte-sized reads to byte-addresses). However, the memory reference misses in the L1-cache (assume that the L1-cache is intially empty). Immediately afterwards, the application accesses main memory, where it experiences a row-buffer miss (initially, assume that all banks in main memory each have a row opened that will never be accessed by the application). Eventually, the cache block (and only that cache block) that contains the byte-address **0xffbc67** is fetched from memory into the cache.

Subsequent memory references may experience *bank-conflicts* in the L1-cache and/or main memory, if there is a previous reference still being served at that particular bank. Bank-conflicts are denoted as hatched shapes in the timeline.

The following table shows the address of the memory references made by the application in both hexadecimal and binary representations. (The table has been replicated for your benefit.)

(a) Memory address table	(b) Same table (for use as scratchpad)
Hexadecimal	Binary	Hexadecimal Binary
ffbc67	1111 1111 1011 1100 0110 0111	ffbc67 1111 1111 1011 1100 0110 0111
ffbda7	1111 1111 1011 1101 1010 0111	ffbda7 1111 1111 1011 1101 1010 0111
ffbdd7	1111 1111 1011 1101 1101 0111	ffbdd7 1111 1111 1011 1101 1101 0111
ff5e28	1111 1111 0101 1110 0010 1000	ff5e28 1111 1111 0101 1110 0010 1000
ff5a28	1111 1111 0101 1010 0010 1000	ff5a28 1111 1111 0101 1010 0010 1000
ff4e28	1111 1111 0100 1110 0010 1000	ff4e28 1111 1111 0100 1110 0010 1000
ff4c04	1111 1111 0100 1100 0000 0100	ff4c04 1111 1111 0100 1100 0000 0100
ffbc6f	1111 1111 1011 1100 0110 1111	ffbc6f 1111 1111 1011 1100 0110 1111
ffbc70	1111 1111 1011 1100 0111 0000	ffbc70 1111 1111 1011 1100 0111 0000

From the above timelines and the table, your job is to answer questions about the processor's cache and main memory organization. Here are some assumptions to help you along the way.

- Assumptions about the L1-cache
 - Block size: ? (Power of two, greater than two)
 - Associativity: ? (Power of two, greater than two)
 - Total data-store size: ? (Power of two, greater than two)
 - Number of banks: ? (Power of two, greater than two)
 - Initially empty

- Assumptions about main memory
 - Number of channels: 1
 - Number of ranks per channel: 1
 - Number of banks per rank: ? (Power of two, greater than two)
 - Number of rows per bank: ? (Power of two, greater than two)
 - Number of cache-blocks per row: ? (Power of two, greater than two)
 - Contains the entire working set of the application
 - Initially, all banks have their 0^{th} row open, which is never accessed by the application

1. First, let's cover the basics

a) Caches and main memory are sub-divided into multiple banks in order to allow parallel access. What is an alternative way of allowing parallel access?

Multiporting, duplicating

b) A cache that allows multiple cache misses to be outstanding to main memory at the same time is called what? (Two words or less. Hint: It's an adjective.)

Non-blocking (or lockup-free)

c) While cache misses are outstanding to main memory, what is the structure that keeps bookkeeping information about the outstanding cache misses? This structure often augments the cache.

Miss status handling registers (MSHRs)

d) Which is larger, an SRAM cell or a DRAM cell?

SRAM cell

e) What is the number of transistors and/or capacitors needed to implement each cell, including access transistor(s)?

SRAM: 6T

DRAM: 1T-1C

2. Cache and memory organization

NOTE: For the following questions, assume that all offsets and indexes come from contiguous address bits.

a) What is the L1-cache's block size in bytes? Which bit positions in the 24-bit physical address correspond to the cache block offset? (The least-significant bit in the physical address has a bit position of 0.)

Block size: 16 bytes

Bit positions of block offset: 0-3

b) How many banks are there in the L1-cache? Which bit positions in the 24-bit physical address correspond to the L1-cache bank index? (The least-significant bit in the physical address has a bit position of 0.)

Number of L1-cache banks: 4

Bit positions of L1-cache bank index: 4-5

c) How many banks are there in main memory? Which bit positions in the 24-bit physical address correspond to the main memory bank index? (The least-significant bit in the physical address has a bit position of 0.)

Number of main memory banks: 8

Bit positions of main memory bank index: 10-12

d) What kind of interleaving is used to map physical addresses to main memory?

Row-interleaving

e) To fully support a 24-bit physical address space, how many rows must each main memory bank have? Which bit positions in the 24-bit physical address correspond to the main memory row index? (The least-significant bit in the physical address has a bit position of 0.)

Number of rows per main memory bank: 2048

Bit positions of row index: 13-23

f) Each cache block within a row is called a *column*. How many columns are there in a single row? Which bit positions in the 24-bit physical address correspond to the main memory column index? (The least-significant bit in the physical address has a bit position of 0.)

Number of columns per row: 64

Bit positions of column index: 4-9

VI. Memory Scheduling (32 points)

To serve a memory request, the memory controller issues one or multiple DRAM commands to access data from a bank. There are four different DRAM commands.

- ACTIVATE: Loads the row (that needs to be accessed) into the bank's row-buffer. This is called *opening* a row. (Latency: 15ns)
- PRECHARGE: Restores the contents of the bank's row-buffer back into the row. This is called *closing* a row. (Latency: 15ns)
- READ/WRITE: Accesses data from the row-buffer. (Latency: 15ns)

The following figure shows the snapshot of the memory request buffers (in the memory controller) at t_0 . Each request is color-coded to denote the application to which it belongs (assume that all applications are running on separate cores). Additionally, each request is annotated with the address (or index) of the row that the request needs to access (e.g., $R\beta$ means that the request is to the 3^{rd} row). Furthermore, assume that all requests are read requests.



A memory request is considered to be *served* when the **READ** command is complete (i.e., 15ns after the request's **READ** command has been issued). In addition, each application (A, B, or C) is considered to be **stalled** until *all* of its memory requests (across all the request buffers) have been served.

Assume that, initially (at t_0) each bank has the 3^{rd} and the 12^{th} row loaded in the row-buffer, respectively. Furthermore, no additional requests from any of the applications arrive at the memory controller.

1. Application-Unaware Scheduling Policies

a) Using the FCFS scheduling policy, what is the stall time of each application?

App A: MAX(2H+7M, H+9M) = H+9M = 15+405 = 420ns

App B: MAX(2H+8M, H+8M) = 2H+8M = 30+360 = 390ns

App C: MAX(2H+9M, H+10M) = H+10M = 15+450 = 465ns

b) Using the **FR-FCFS** scheduling policy, what is the stall time of each application? (The figure is replicated and provided below for your benefit.)

App A: MAX(5H+2M, (4H+2M)+4M) = 4H+6M = 60+270 = 330ns

App B: MAX((5H+2M)+3M, 4H+2M) = 5H+5M = 75+225 = 300ns

App C: MAX(((5H+2M)+3M)+M, ((4H+2M)+4M)+M) = 4H+7M = 60 + 315 = 375ns



c) What property of memory references does the FR-FCFS scheduling policy exploit? (Three words or less.)

Row-buffer locality

d) Briefly describe the scheduling policy that would **maximize** the *request throughput* at any given bank, where request throughput is defined as the number of requests served per unit amount of time. (Less than 10 words.)

FR-FCFS

2. Application-Aware Scheduling Policies

Of the three applications, application C is the least memory-intensive (i.e., has the lowest number of outstanding requests). However, it experiences the largest stall time since its requests are served only after the numerous requests from other applications are first served. To ensure the shortest stall time for application C, one can assign its requests with the highest priority, while assigning the same low priority to the other two applications (A and B).

a) Scheduling Policy X: When application C is assigned a high priority and the other two applications are assigned the same low priority, what is the stall time of each application? (Among requests with the same priority, assume that FR-FCFS is used to break ties.) The figure is replicated and provided below for your benefit.

App A: MAX(M+(4H+3M), M+(3H+3M)+4M) = 3H+8M = 45+360 = 405ns

App B: MAX(M+(4H+3M)+3M, M+(3H+3M)) = 4H+7M = 60+315 = 375ns

App C: MAX(M, M) = M = 45ns



Can you design an even better scheduling policy? While application C now experiences low stall time, you notice that the other two applications (A and B) are still delaying each other.

b) Assign priorities to the other two applications such that you minimize the average stall time across all applications. Specifically, list all **three** applications in the order of highest to lowest priority. (Among requests with the same priority, assume that FR-FCFS is used to break ties.)

C > B > A

c) Scheduling Policy Y: Using your new scheduling policy, what is the stall time of each application? (Among requests with the same priority, assume that FR-FCFS is used to break ties.) The figure is replicated and provided below for your benefit.

App A: MAX(M+(3M)+(4H+3M), M+(3H+3M)+4M) = 3H+8M = 45+360 = 405nsApp B: MAX(M+(3M), M+(3H+3M)) = 3H+4M = 45+180 = 225ns

App C: MAX(M, M) = M = 45ns



d) Order the four scheduling policies (FCFS, FR-FCFS, X, Y) in the order of lowest to highest average stall time.
Y<X<FR-FCFS<FCFS</p>

VII. Tomasulo's Algorithm Strikes Back (25 points)

The diagram below shows a snapshot at a particular point in time of various parts of the microarchitecture for an implementation supporting out-of-order execution in the spirit of Tomasulo's Algorithm.

Note that both the adder and multiplier are pipelined, and adder has two stages and the multiplier has five stages. At the particular point in time when the snapshot was taken, the adder had no instructions in the pipeline and the multiplier had one instruction in the pipeline (in stage four). Note that the adder and multiplier have separate output data buses, which allow both an add result and a multiply result to update in the same cycle.

The adder has two reservation stations and the multiplier has four reservation stations. An instruction continues to occupy a reservation station slot until it completes execution and its result is written to its destination register.

The processor has been supplied with a five-instruction program segment. Instructions are fetched, decoded, and executed as discussed in class. Assume full data forwarding, whenever possible.

Reg	V	Tag	Value
r0	1	q	5
r1	0	r	3
r2	1	У	7
r3	0	n	37
r4	0	d	4
r5	0	х	6
r6	1	Z	32
r7	1	0	23

ADD Reservation Stations										
DestReg	V	Tag	Value	V	Tag	Value				
d	0	х	_	1	с	37				
—	—	1	_	-	_	_				

MUL Reservation Stations											
DestReg	V	Tag	Value	V	Tag	Value					
r	0	n	_	0	d	-					
n	0	d	_	1	с	37					
х	1	s	3	1	с	37					
_	—		_	—	_	_					

ADD Pipe	line
Stage 1	-
Stage 2	-

MUL Pipe	line
Stage 1	I
Stage 2	-
Stage 3	-
Stage 4	х
Stage 5	

1. Data Flow Graph

Identify the instructions and draw the data flow graph for the five instructions (use + for ADD and \times for MUL). Please label the edges of the dataflow graph with the destination register tag. Note that one instruction has already completed and written back its result, ADD r3, r0, r6.



2. Execution Timeline

a) Fill in the instruction opcodes, source, and destination registers. (Please put the source registers in numerical order.)

rabie r. mon denoms											
Opcode	Dest	Src1	Src2								
ADD	r3	r0	r6								
MUL	r5	r1	r3								
ADD	r4	r3	r5								
MUL	r3	r3	r4								
MUL	r1	r3	r4								

Table 1: Instructions

b) Fill in the execution timeline using the letters F for fetch stage, D for decode stage, E for an execute stage, R for update reorder buffer stage, and W for register write stage. Use a dash (-) to denote a stall.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
F	D	Е	Е	R	W																		
	F	D	-	Е	Ε	Е	Е	Е	R	W													
		F	D	-	-	-	-	-	Е	Ε	R	W											
			F	D	-	-	-	-	-	-	Ε	Е	Ε	Е	Ε	R	W						
				F	D	-	-	-	-	-	-	-	-	-	-	Ε	E	Ε	Ε	Ε	R	W	

Table 2: Execution timeline

c) From the above execution timeline, in which cycle was the snapshot was taken?

Cycle 8.