Last Name: SAMPLE. First Name: ____________________

Student ID: ____________________ Signature: ____________________

**Course 304-425B -- Computer Organization and Architecture**

Final examination

December 21, 2001, 14:00 -- 17:00

Examiner: Prof. V. Hayward

Associate Examiner: Prof. J. Ferrie

## INSTRUCTIONS

□ **This is a closed book examination.** Calculators and one or two sheets of notes are allowed.

□ Explain every result concisely **when asked.** Marks will be given for clear, concise solutions.

□ State any assumption required for an answer if it is not clear in the text of the question.

□ This exam has 14 pages including this one. It has 7 sections for 25 questions. Each question is indicated by a bullet sign (•) and carries four (4) marks. The marks add up to 100.

□ Please sign this paper at the top of the page, write your name and student number legibly there.

□ **Put your answers in the space provided** and keep all the pages together.

### PLEASE NOTE CAREFULLY

□ Make sure that the signed paper in its entirety is handed in (along with all signed exam books) at the end of the examination.

□ Make sure that the answers are put in the space provided, **answers in any other location will not be marked.**

□ You have approximately 180 minutes to complete the exam.

## PROBLEMS
## SECTION 1: Performance (5 questions)

- Consider adding an enhancement to a machine that would divide the number of load/store instructions by two. All other instructions would be left unchanged. However, the clock cycle of the machine would be 7% longer and the average CPI for all instructions would change from 1.1 clock cycle, before enhancement, to 1.2 after enhancement. What is the minimum percentage of load/stores in the instruction mix produced by an application would be required to make the enhancement worth implementing, assuming it is free of cost?

$$\text{TIME NEW} \leq \text{TIME OLD} \Rightarrow IC\ 1.1\ CC \leq IC\ 1.2\left(1-\frac{\alpha}{2}\right) 1.07\,CC$$

$$1.1 \leq 1.2\left(1-\frac{\alpha}{2}\right)1.07 \Rightarrow \frac{\alpha}{2} \geq \left(1-\frac{1.1}{1.2\ 1.07}\right)$$

$$\Rightarrow \alpha \geq 0.286$$

- Consider now a five stage pipeline which causes 20% of instructions to stall for one cycle on average. Consider the memory to be perfect, that is, there is no memory stalls. Assuming that the clock rate is four times faster than a non-pipelined version of the same CPU, what is the effective improvement in performance?

ONE STAGE $\quad \text{TIME} = IC\ CPI\ CC$

$$\text{TIME} = IC\ \frac{CPI}{5}\ \frac{CC}{4}\ 1.2 \qquad SU = \frac{1}{\frac{1}{5}\frac{1}{4}1.2} = 16.7$$

5 INSTRUCTIONS OVERLAP

CLOCK CYCLE TIME REDUCED

- Consider a single pipeline machine like DLX. For a given benchmark suite, measurements show that 10% of the instructions are unconditional jumps. A first enhancement is proposed in the form of a "jump target buffer" and a second in the form of "jump folding". Remember that the jump target address is calculated in the ID stage and the buffer updated in the EX stage. For these enhancement, the hit rate is 90%. The average CPI of the base machine is estimated to be 1.5 for all other instructions. All other aspects of the machine are assumed to be unchanged. Compute the expected speed-up due to the implementation of:

◇ a "jump target buffer" NORMAL DLX $CPI\_OLD = 0.1 * 2 + 0.9 * 1.5 = 1.55$

$$CPI\_NEW = 0.1\,(0.9 * 1 + 0.1 \times 2) + 0.9 * 1.5$$
$$= 0.11 + 1.35 = 1.46$$

$$SU = \frac{1.55}{1.46} = 1.061$$

◇ "jump folding"

$$CPI\_NEW = 0.1\,(0 + 0.1 \times 2) + 0.9 * 1.5$$
$$= 0.02 + 1.35 = 1.37$$

$$SU = \frac{1.55}{1.37} = 1.13$$

- Consider now a machine with a virtual memory and two levels of cache. The performance figures are:

L1: the hit time is 1 cc and the miss rate is 5%
L2: the hit time is 10 cc and the miss rate is 2%
MM: the hit time for a block is 40 cc, the miss rate 0.0001%, and miss penalty is 10,000,000 cc

Compute the global average memory access time (AMAT)

$AMAT_i = HT_i + MR_i \; MP_i$ $\qquad MP_i = AMAT_{i+1}$

AT ANY LEVEL

MISS PENALTY DETERMINE BY ACCESS TO HIGHER LEVEL.

$$AMAT = 1 + 0.05 \; (10 + 0.02 \; (40 + 10^{-6} \; 10^{7}))$$
$$= 1 + 0.05 \; (10 + 0.02 \; 50)$$
$$= 1 + 0.05 \; 11 = 1.55$$

- We focus now on the performance of a DLX style machine with one level of cache that has a 1 cc hit time, a 5% miss rate, and a miss penalty of 16 cc. We improve this cache so it becomes non blocking and is such that the CPU can restart as soon as the first word arrives from the MM 4 cc after request. This improvement concerns the instruction fetches only. The data cache remains blocking and 30% of all instructions are loads. The data cache is "write through" but has a write buffer that eliminates data stalls for stores. What is the speed up introduced by the non blocking feature?

$CPI_{AVE} =$ CPI_BASE + FETCH-MISS-RATE * PENALTY + FREQ-L/S * DATA MISS-RATE * PENALTY

BLOCKING CASE

$$CPI_{AVE} = 1.0 + 0.05 \; 16 + 0.3 \; 0.05 \; 16 = 2.04$$

NON BLOCKING CASE

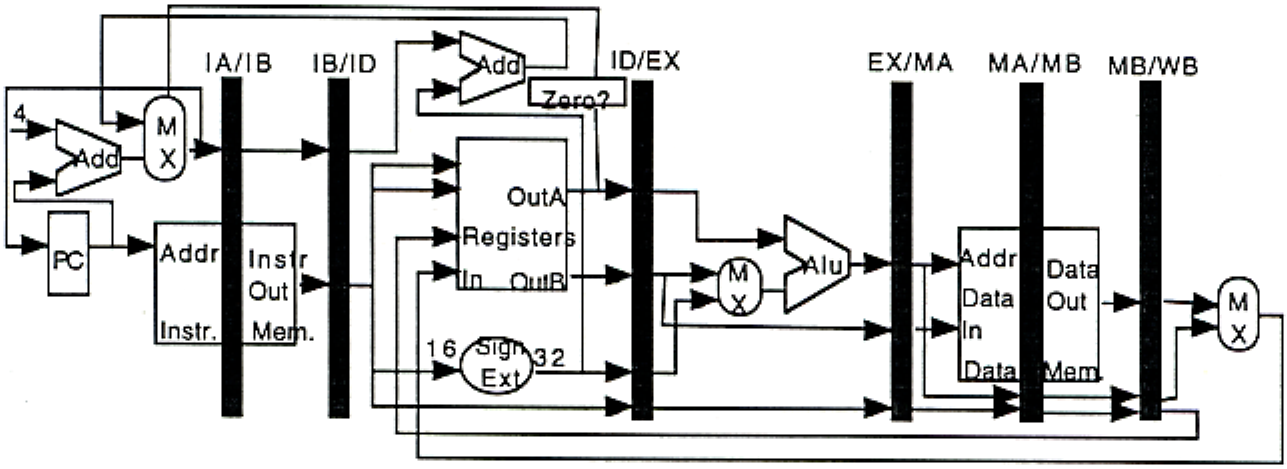
$$CPI_{AVE} = 1.0 + 0.05 \; 4 + 0.3 \; 0.05 \; 16 = 1.44$$

$$SU = \frac{2.04}{1.44} = 1.42$$

END OF SECTION 1 (5 questions)

## SECTION 2: Integer Pipelining (4 questions)

Timing analysis reveals that the memory cycles in the standard DLX pipeline are the limiting factor for clock cycle time improvement. The machine is super-pipelined. Complete instruction fetches takes two stages IA and IB. In the first stage, the memory addressed is specified, in the second, the instruction is read out. The same technique is applied to the MEM stage, now split into a MA and a MB stage. The new design is fully pipelined but has This is symbolically represented by indicating new pipe registers.



**SUPERPIPELINED DLX**

- We need a sequence of pipelined DLX code to compute: $\begin{cases} 2a+1-c & \text{if } 2a+1 \geq 0 \\ 2b+1-2c & \text{otherwise} \end{cases}$

Upon entry, *a* is stored in register R1 and the result is left in register R1, *b* is in R2, *c* is in R3. The pipeline handles all structural and data hazards. The numbers processed are expected to be equally likely to yield one case or the other. Control hazards are handled via "delayed branch with canceling". Here is a semi-optimized sequence:

```
        ADD    R4, R1, R1    2a
        ADDI   R4, R4, 1     2a+1
        SGE    R5, R4, R0    <0 ?
        BNEZ   R5, SKIP
        ADD    R3, R3, R3    2c
        ADD    R4, R2, R2    2b
        ADDI   R4, R4, 1     2b+1
SKIP    SUB    R1, R4, R3
```

Use speculative execution to write an optimized sequence. The machine does not have hardware speculative features such as boosted instructions or poison bits. (Hint: do not try to manipulate the expressions compute them as given).

DELAY SLOT HAS TWO PLACES.
MOVE CONTROL DEPENDENT INSTRUCTION BEFORE BRANCH TO SPECULATIVELY CALCULATE 2b, FOR EXAMPLE.

```
       ADD    R4, R1, R1
       ADDI   R4, R4, 1      2a+1
       SGE    R5, R4, R0
       ADD    R6, R2, R2     RENAME 2b
       BNEZ   R5, SKIP
       ADD    R3, R3, R3     2c
       ADDI   R4, R6, 1      2b+1
SKIP:  SUB    R1, R4, R3
```

ANOTHER POSSIBILITY

```
       ADD    R4, R1, R1
       ADD    R4, R4, 1      2a+1
       SGE    R5, R4, R0
       SUB    R1, R4, R2     2a+1-C    ?
       BNEZ   R5, SKIP
       ADD    R3, R3, R3     2c
       ADD    R4, R2, R2     2b
       ADDI   R4, R4, 1      2b+1-2C
       SUB    R1, R4, R2
SKIP: OTHER CODE
```

- Rewrite the untransformed sequence above, assuming now that the pipeline has no bypassing and no canceling hardware at all, inserting NOPs (no operation instructions) to ensure correct execution.

3 CASES

```
ALU → ALU    IA  IB  ID  EX  MA  MB  WB
                 X   X   X   X   IA  IB  ID

ALU → BR     IA  IB  ID  EX  MA  MB  WB
                 X   X   X   X   IA  IB  ID
                                         TEST

BR DELAY     IA  IB  ID  -  -  -
                 X   X   IA  IB
```

```
NOPS
    ADD    R4, R1, R1
4 >
    ADDI   R4, R4, 1
4 >
    SGE    R5, R4, R0
4 >
    BNEZ   R5, SKIP
2 >
    ADD    R3, R3, R3
    ADD    R4, R2, R2
4 >
    ADDI   R4, R4, 1
4 >
    SUB    R1, R4, R3
```

- We now consider the CPU to be fully bypassed and to handle branches with pure delayed branch. Write an optimized sequence that executes correctly.

NOMINAL SEQUENCE
BRANCH DELAYED

```
       ADD    R4, R1, R1
       ADDI   R4, R4, 1     2a+1
       SGE    R5, R4, R0
       BNEZ   R5, SKIP
        NOP
        NOP
       ADD    R3, R3, R3    2c
       ADD    R4, R2, R2    2b
       ADDI   R4, R4, 1     2b+1
SKIP:  SUB    R1, R4, R3
```

NOTHING TO MOVE IN SLOT FROM BEFORE BRANCH. BRING INSTRUCTION FROM FALL THROUGH SEQUENCE IF MISTAKEN EXEC OK. MUST CREATE A "TEMP"

```
ADD    R4, R1, R1
ADDI   R4, R4, 1
SGE    R5, R4, R0
BNEZ   R5, SKIP
ADD    R6, R2, R2    2b IN TEMP
NOP
ADD    R3, R3, R3
ADDI   R4, R6, 1
SUB    R1, R4, R3
```

ANOTHER POSSIBILITY

```
      ADD    R4, R1, R1
      ADDI   R4, R4, 1
      SGE    R5, R4, R0
      BNEZ   R5, SKIP
      SUB    R1, R4, R3
      ADD    R3, R3, R3
      ADD    R4, R2, R2
      ADDI   R4, R4, 1
      SUB    R1, R4, R3

SKIP  OTHER CODE
```

- For the pipeline diagramed above with full bypassing, including around the MA MB stages, above what is the load latency for these two cases?

◇
```
LW    R1, 0(R2)      EX  MA  MB
ADD   R1, R1, R1         X   X   EX          2 CYCLE
```

◇
```
LW    R1, 8(R2)      EX  MA  MB
SW    R1, 0(R3)          X   EX  MA  MB      1 CYCLE
```

END OF SECTION 2 (4 questions, 9 so-far)

## SECTION 3: Loop Transformations With FP Code (3 questions)

The problem is evaluate the benefit of loop unrolling. The FPUs are fully pipelined and bypassed. However the FP division is not pipelined. In case of write contention (one write per clock cycle) the earliest instruction has priority and stalls the contending instruction(s). Ignore the branch delay and assume the following data:

| Instruction producing result | Instruction using result | Latency in clock cycles |
|---|---|---|
| Load double | any FPU operation | 1 |
| completion of any FPU operations | Store Double | 0 |
| FP ADD | FPU operation | 3 |
| FP MULT | FPU operation | 4 |
| FP DIV | FPU operation | 15 |

```
LOOP: LD    F2, 0(R1)          A[I] = A[I] * c
      MULTD F4, F2, F0
      SD    F4, 0(R1)
      LD    F4, 0(R2)          B[I] = (B[I] + B[I]) / c
      ADDD  F6, F4, F4
      DIVD  F6, F6, F0
      SD    F6, 0(R2)
      ADDI  R1, R1, 8          INTEGER
      ADDI  R2, R2, 8          INDEX AND
      ADDI  R5, R5, -1         LOOP
      BNEZ  R3, LOOP           CONTROL
```

- Unroll the loop twice, and use other transformation to reduce stalls. For conciseness, ignore the integer code and schedule the FP instruction as if it did not exist.

OBSERVATIONS: 2 INDEPENDENT CALCULATIONS
DIVD NOT PIPELINED, BEST WE CAN HOPE IS $\frac{30}{2}$ CC, IE 2 DIVD

BASIC LOOP

```
   LD     A[I]
    1
8  MULTD  A[I] * C
    4
   SD     A[I]
   LD     B[I]
    1
   ADDD   B[I]+B[I]
23  3
   DIVD   ( ) / c
    15
   SD     B[I]
31 cc
```

IN OTHER WORDS,
7 FETCHES AND
24 STALLS

STRATEGY: START WITH B AND SCHEDULE SECOND LOOP IN THE STALLS. LOTS OF POSSIBILITIES, BUT THE DIVDS CANNOT OVERLAP.

```
    LD    F4, 0(R2)      B[I]
    LD    F10, 8(R2)     B[I+1]
7   ADDD  F6, F4, F4
    ADDD  F16, F10, F10
    LD    F2, 0(R1)      A[I]
    LD    F12, 8(R1)     A[I+1]
    DIVD  F14, F6, F0
    MULTD F4, F2, F0
    MULTD F10, F12, F0
15  SD    F6, 0(R2)
    SD    F16, 8(R2)
    DIVD  F20, F16, F0
15  SD    F14, 0(R2)
    SD    F20, 8(R2)
```

CAN SOFTWARE PIPELINE
THAT VERSION OR EVEN ORIGINAL ONE

```
LD    F4, 0(R2)      B[I]
LD    F10, 8(R2)       I+1
ADDD  F6, F4, F4
ADDD  F16, F10, F10
LD    F2, 0(R1)      A[I]
LD    F12, 8(R1)     A[I+1]
SD    F14, -16(R2)   B[I-2]
DIVD  F14, F6, F0
MULTD F4, F2, F0
MULTD F4, F12, F0
SD    F6, 0(R2)
SD    F16, 8(R2)
SD    F20, -8(R2)    B[I-1]
DIVD  F20, F16, F0
```

- Calculate the number of clock cycles to perform the basic computation, comparing the untransformed loop with the unrolled one:

47 CYCLES
GIVE OR TAKE
A FEW

30 CYCLES
FOR TWO LOOPS.

- Suppose now that the machine is a super-scalar, dual issue machine with two pipelines: one to process integer, branches and all load/stores instructions, and the other to process all the FP instructions.

  ◇ Use the chart below to show a schedule that maximizes throughput of one single loop accounting for integer code. Assuming that the branch is folded and that there is a hit. You may use more registers if needed.

| | Integer Instruction | FP Instruction |
|---|---|---|
| CC1 LOOP | LD F8, 0(R2) | |
| CC2 | LD F2, 0(R1) | |
| CC3 | ADDI R1, R1, 8 | ADDD F6, F8, F8 |
| CC4 | ADDI R2, R2, 8 | MULTD F4, F2, F0 |
| CC5 | ADDI R5, R5, -1 | DIVD F6, F6, F0 |
| CC6 | SD F4, -8(R1) | |
| CC7 | SD F6, -8(R2) | |
| CC8 | BNEZ R5, LOOP | |
| ~~CC9~~ | | |
| CC10 | SD F6, -8(R2) | ADD F6, F8, F8 |
| CC11 | SD F4, -8(R1) | MULTD F4, F2, F0 |
| CC12 | LD F8, 0(R2) | DIVD F6, F6, F0 |
| CC13 | LD F2, 0(R1) | |
| CC14 | ADDI R5, R5, -1 | |
| CC15 | ADDI R1, R1, 1 | |
| CC16 | ADDI R2, R2, 1 | |
| CC17 | BNEZ R5, SKIP | |
| CC18 | | |
| CC19 | | |
| CC20 | WITH SOFT PIPELINING | |
| CC21 | | |
| CC22 | | |
| CC23 | | |
| CC24 | | |
| CC25 | | |
| CC26 | | |
| CC27 | | |
| CC29 | | |
| CC30 | | |

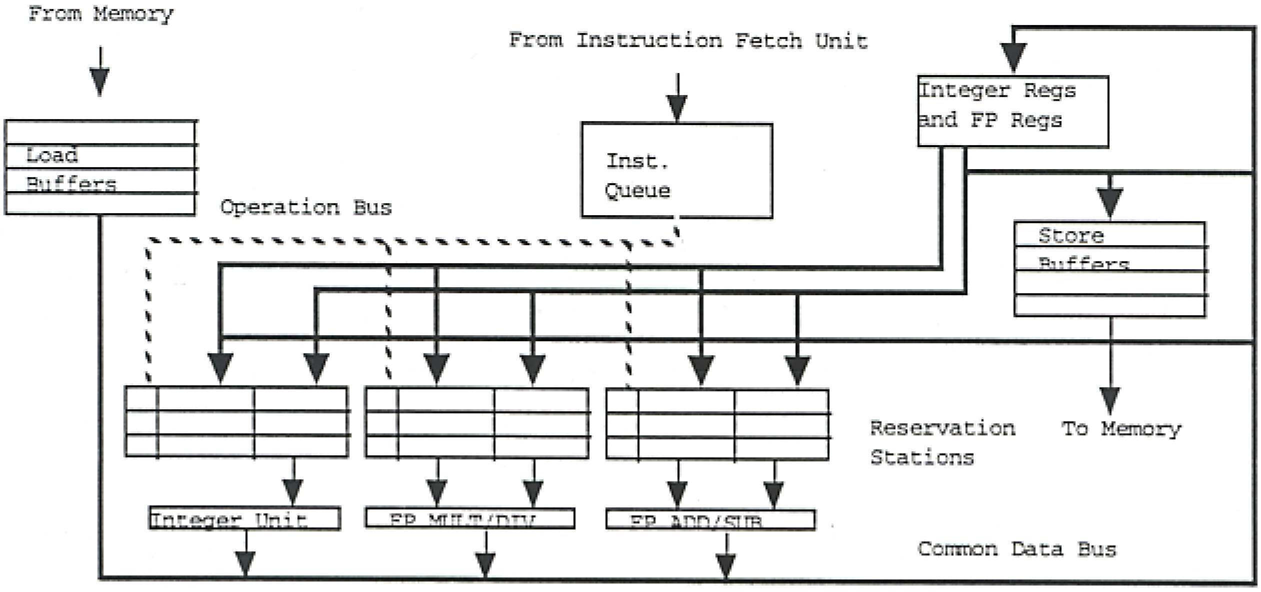
ASSUMING THAT HAZARD DETECTION AUTOMATICALLY DELAYS EXECS

◇ Explain in a few words the transformation(s) that you have used:

- RENAMED F4 TO F8 TO AVOID OUTPUT DEPENDENCY
- PERFORME DIVD AS EARLY AS POSSIBLE TO ALLOW FOR MORE OVERLAP

## SECTION 4:Tomasulo's Algorithm (4 questions)

Consider the pipeline below. The integer unit can be controlled to carry out any type of integer instructions. It has one FP ADD/SUB unit and one MULT/DIV unit. The load and store buffers have four entries each. The reservation stations have three entries.



Assume that the latencies are 0 for all integer operations, 1 for loads, 4 for the FP add/sub's, 6 for the multiplication and 25 for the divides, regardless of the instruction using the result. All units are fully pipelined. The Common Data Bus is written at the end of the last clock cycle of an operation. An operation that depends on the written value starts at the next clock cycle. The Common Data bus cannot support multiple data transfers. In case of contention for the bus, the i~~nstruction~~ the earliest issued instruction has priority.

Consider now the following sequence of code assuming that a new instruction is fetched at each clock cycle and issued at the next clock cycle if there is free entry in the reservations or in the buffers. A table to indicate timing is partially filled.

- Complete this table first:

| fetched | Instr. | Operands | Issue | Exec | Write |
|---|---|---|---|---|---|
| CC1 | LD | F2, 0(R1) | CC2 | CC3-4 | CC4 |
| CC2 | LD | F4, 4(R1) | CC3 | CC4-5 | CC5 |
| CC3 | MULTD | F2, F2, F4 | CC4 | CC6-12 | CC12 |
| CC4 | DIVD | F4, F2, F6 | CC5 | CC13 - CC38 | CC 38 |
| CC5 | SUBD | F6, F4, F2 | CC6 | CC39 - CC43 | CC 43 |
| CC6 | ADDD | F6, F2, F2 | CC7 | CC13 - CC17 | CC17 |
| CC7 | ADDI | R1, R1, 8 | CC8 | CC9 | CC9 |
| CC8 | SD | F4,8(R1) | CC9 | CC 39 | NEVER |
| | | | | | |
| | | | | | |

AT CC7
LOADS COMPLETED
MULTD EXECUTES
DIVD ISSUED, WAITING FOR MULTD
SUBD ISSUED, WAITING FOR DIVD AND MULTD
ADDD ISSUED, WAITING FOR MULTD

CC 14
MULTD COMPLETED
DIVD EXECUTES
SUBD ISSUED, WAITING FOR DIVD
ADDD EXECUTES
ADDI COMPLETED
SD ISSUED, WAITING FOR DIVD

Use "`Mem[Reg[R1]]`" to denote, for example, the *value* fetched by the first load, "`Reg[R1]`" to denote the *value* held in register `R1`, and `#8` to denote the value 8.

- Indicate in the tables below the state of the reservation stations and of the registers during clock cycle 7.

| Name | Busy | Op. | Vj | Vk | Qj | Qk |
|---|---|---|---|---|---|---|
| MULT/DIV-1 | ✓ | MULTD | MEM[REG[R1]] | MEM[REG[R1]+4] | | |
| MULT/DIV-2 | ✓ | DIVD | | REG [F6] | MULT/DIV-1 | |
| MULT/DIV-3 | | | | | | |
| ADD/SUB-1 | ✓ | SUBD | | | MULT/DIV-2 | MULT/DIV-1 |
| ADD/SUB-2 | ✓ | ADDD | | | MULT/DIV-1 | MULT/DIV-1 |
| ADD/SUB-3 | | | | | | |
| INT-1 | | | | | | |
| INT-2 | | | | | | |
| INT-3 | | | | | | |

| Field | F0 | F2 | F4 | F6 | F8 | R1 | R2 | R3 | R4 |
|---|---|---|---|---|---|---|---|---|---|
| Qi | | MULT/DIV-1 | MULT/DIV-2 | ADD/SUB-2 | | | | | |

- Indicate in the table below the state of the reservation stations and of the store buffers during clock cycle 14.

| Name | Busy | Op. | Vj | Vk | Qj | Qk |
|---|---|---|---|---|---|---|
| MULT/DIV-1 | | | | | | |
| MULT/DIV-2 | ✓ | DIVD | REG [F2] | REG [F6] | | |
| MULT/DIV-3 | | | | | | |
| ADD/SUB-1 | | SUBD | | REG [F2] | MULT/DIV-2 | |
| ADD/SUB-2 | | ADDD | REG [F2]. | REG [F2] | | |
| ADD/SUB-3 | | | | | | |
| INT-1 | | | | | | |
| INT-2 | | | | | | |
| INT-3 | | | | | | |

| Field | Store-1 | Store-2 | Store-3 | Store-4 |
|---|---|---|---|---|
| Qi | MULT/DIV-2 | | | |
| Busy | ✓ | | | |
| Address | 8+ REG [R1] | | | |

- Explain in 20 words or less the motivation and the function of a reorder buffer:

- MAKES IT POSSIBLE TO EXCUTE INSTRUCTIONS SPECULATIVELY, I.E. BEFORE A BRANCH OUTCOME IS KNOWN, HOLDING RESULTS BEFORE WRITTING THEM.
- (INSTRUCTIONS HAVE AN EXTRA "COMMIT" STEP).

END OF SECTION 4 (4 questions, 16 so-far)

## SECTION 5: Branch predictors (2 questions)

Consider this nested loop:

```
for (i = 0; i < 100; ++i) ———————————— 100
    for (j = 0; j < 10; ++j)——————————— 1000
        if (j % 2 == 0) ——————————————— 1000
            if (i % 2 == 0) —— 500
                a[j] = 0;
        else
            a[j] = 1;
```

- A machine has a 2-bit branch predictor. Estimate how many correct and how many incorrect predictions are made while executing this code, assuming a 100% hit rate and no clashes in the predictor table (Hint: start with the inner most code and work your answer step by step, giving just a number will earn you no marks).

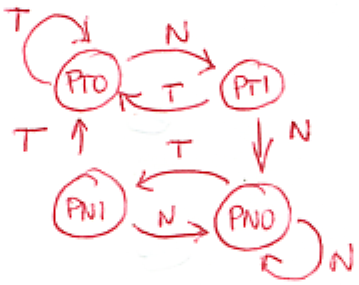
| | MISPREDICTIONS | CORRECT PREDICTIONS |
|---|---|---|
| INNER "IF" VISITED 500 TIMES WITH PATTERN TTTTT NNNNN T... | 100 | 400 |
| FIRST "IF" VISITED 1000 TIMES WITH PATTERN TNTN--- | 500 | 500 |
| INNER "FOR" VISITED 1000 TIMES EXITS 100 TIMES | 100 | 900 |
| OUTER "FOR" VISITED 100 TIMES EXITS ONCE | 1 | 99 |
| 2600 VISITS OF WHICH : | 701 | 1899 |
| | MISPREDICTIONS | CORRECT PREDICTIONS |

Recall that a (1,2) predictor refers to a correlating predictor that implements correlation by choosing among two different 2-bit predictors. We consider a certain sequence of code that causes three branches B1, B2, B3 to be visited with the outcomes listed in the table (T taken, N not taken, hint, there is no obvious pattern). Each branch is visited 4 times. Assume that prior entering the sequence, all the predictors are in the state that would result from a series of taken branches and this applies to both predictors of each branch.

| | Branch | B1 | B2 | B3 | B2 | B1 | B3 | B1 | B3 | B2 | B3 | B1 | B2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LAST BRANCH | Outcome | W N | R T | W N | W N | W N | R T | W N | R T | W N | R T | R N | R T |
| TAKEN | Predictor 1 | PT0<br>PT1 ↓ | PT0 | PT0<br>PT1 ↓ | PT0 | PT1 | PT1 | PT1<br>PN0 ↓ | PT1 | PT0<br>PT1 ↓ | PT1 | PN0<br>PN0 ↓ | PT1 |
| NOT TAKEN | Predictor 2 | PT0 | PT0<br>PT0 ↓ | PT0 | PT0<br>PT1 ↓ | PT0<br>PT1 ↓ | PT0<br>PT0 ↓ | PT1 | PT0<br>PT0 ↓ | PT1 | PT0<br>PT0 ↓ | PT1 | PT1<br>PT0 ↓ |

- Work out the number of correct and incorrect predictions for each of the three branches. This can be done by filling in the table above. There is no need to restate the details of 2-bits predictors but nevertheless indicate the functions you assigned to Predictor 1 and to Predictor 2. To clarify your work, use the symbols PT0, PT1, PN0, PN1 to refer to the states of a 2-bit predictor and specify concisely what they mean.

DEF'S



6 RIGHT, 6 WRONG.

END OF SECTION 5 (2 questions, 18 so-far)

## SECTION 6: Loop level parallelism (3 questions)

Consider this loop:

```
for (i = 1; i < 100; ++i) {
    a[i - 1] = c[i - 1] + n;              /* S1 */
    b[i] = m + c[i];                      /* S2 */
    a[i] = a[i] + b[i];                   /* S3 */
}
```

Rewrite the loop so it becomes parallel. Solve this problem in two different ways:

- First use software renaming, not changing the structure of the loop:

LOOP CARRIED OUTPUT DEPENDENCY S1 OVERWRITES S3 VIA a[i], a[i-1]
LOOP CARRIED ANTI-DEPENDENCY S3 MUST READ BEFORE S1 WRITES

HOWEVER a[i-1] ALWAYS WRITTEN, NEVER READ, CALL IT T[i-1]

```
for ( — ) {
    T[i-1] = c[i-1] + n;
        b[i] = m + c[i];
        a[i] = a[i] + b[i];
}
```

EACH ITERATION IS INDEPENDENT FROM THE PREVIOUS ONE

- Second transform the loop without renaming the variables so it becomes parallel:

ONLY ONE DATA DEPENDENCE, S2→S3 START WITH SECOND STATEMENT

```
a[0] = c[0] + n;
for (i = 1; i < 99; ++i) {
        b[i] = m + c[i];
        a[i] = a[i] + b[i];
        a[i] = c[i] + n;
}
b[99] = m + c[99]
a[99] = a[99] + b[99]
```

OR SIMPLY DECID TO ELIMINATE S3

```
for (i = 1, i < 100 ; ++i) {
    a[i-1] = c[i-1] + n;
    b[i] = m + c[i];
}
a[99] = a[99] + b[99]
```

Consider now this loop:

```
for (i = 1; i < 100; ++i) {
        n = a[i] + n;
}
```

- Can it be transformed to become parallel? Explain.

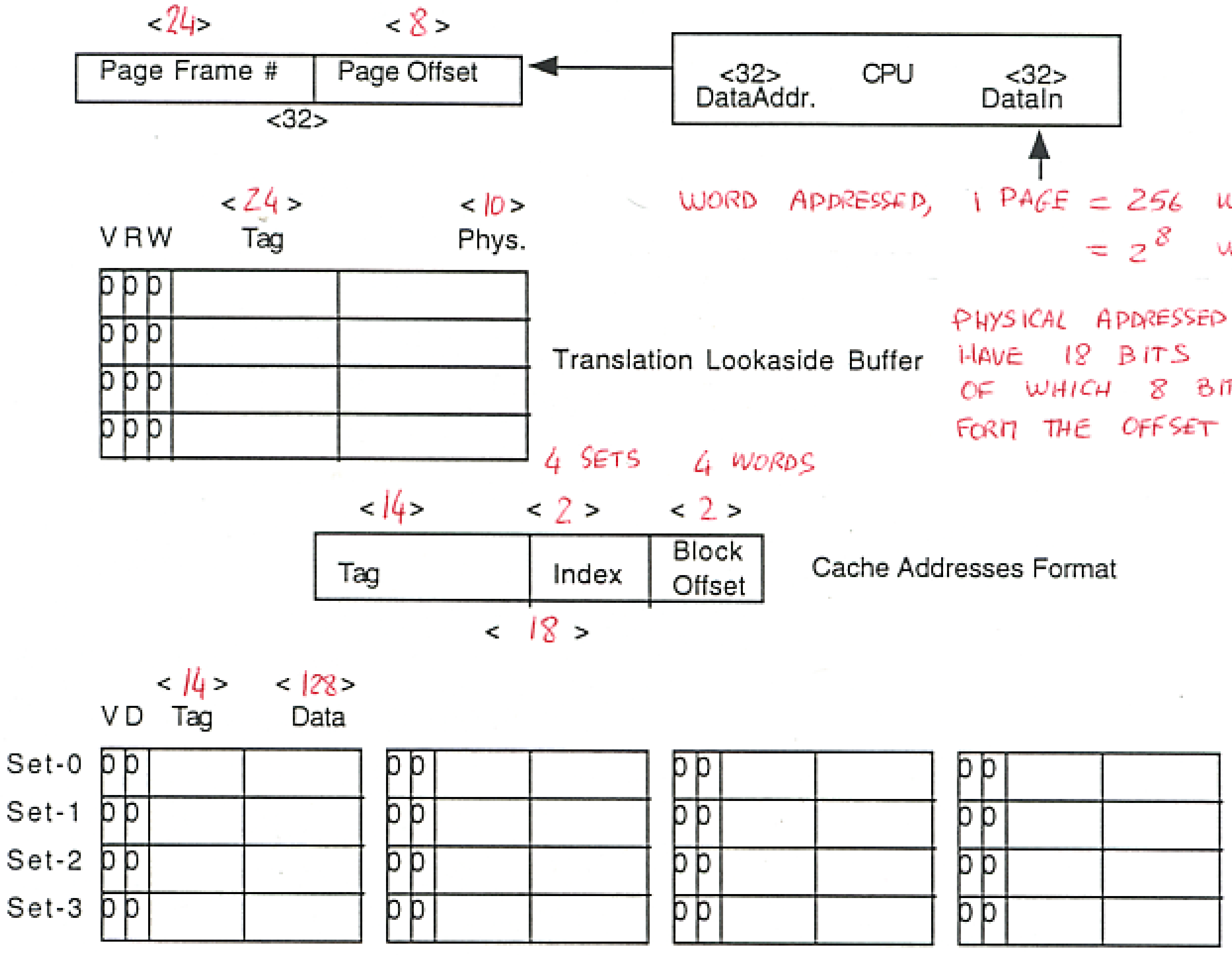
NOMINALLY NO, BECAUSE EACH ITERATION DEPENDS ON THE PREVIOUS ONE.

HOWEVER, SINCE IT'S A RUNNING SUM, WITH PROFOUND TRANSFORMATIONS, PARTIAL SUMS COULD BE COMPUTED IN PARALLEL.

END OF SECTION 6 (3 questions, 21 question so-far)

## SECTION 7: Memory Hierarchy (4 questions)

Consider this hierarchical memory design, along with the following facts: It is word addressed and the words are 32 bit long. The physical memory has a capacity of 1 Megabyte. The cache blocks have 4 words. The page size is 1 kByte



WORD ADDRESSED, 1 PAGE = 256 WORDS
= $2^8$ WORDS

PHYSICAL ADDRESSED HAVE 18 BITS OF WHICH 8 BITS FORM THE OFFSET

Set Associative Cache (4 sets of 4 blocks), Write Back, Write Allocate

- Fill all the missing field size specifications (< >) on the diagram above.

- Compute the total size of the page table: # PAGES * SIZE OF PAGE TABLE ENTRY

SIZE (PTE) = SIZE (PERM BITS) + SIZE (DIRTY BIT) + SIZE (TAG) + SIZE (PHYS. P. #)

= 3 + 1 + 24 + 10

= 48 BITS ≃ 5 BYTES

$$\# \text{ PAGES} = \frac{\text{VIRTUAL ADDR. SPACE}}{\text{PAGE SIZE}} = \frac{2^{32}}{2^8} = 2^{24} = 16 \text{ Megs}$$

SIZE (PAGE TABLE) = 80 Mbytes

- Consider a benchmark such that all the addresses in the address space are sequentially read once for the machine described above. It starts from a blank cache.

◇ What is the number of cache misses?

NO LOCALITY, SO ONE CACHE MISS PER BLOCK

ADDRESS SPACE = $2^{32}$ ; SIZE OF BLOCK = $2^2$ WORDS

$\Rightarrow$ $2^{30}$ CACHE MISSES

◇ What is the number of page faults?

SIMILARLY, PAGE SIZE IS $2^8$ WORDS

$\Rightarrow$ $2^{24}$ PAGE FAULTS

- Answer these yes/no questions

◇ "A victim cache" is a method to reduce the miss rate in caches [yes/no]: YES

◇ Software pipelining is used to minimize the number of page faults [yes/no]: NO (ILP)

◇ "Loop fusion" is a software technique to speed up recursive code [yes/no]: NO (BETTER USE OF CACHE) (ITERATIVE CODE)

◇ "Interleaved Memory" makes it possible to pipeline block replacement in caches [yes/no]: YES

END OF SECTION 7 (4 questions)

AND

END OF EXAM (and end of the 25 questions)