LUND INSTITUTE OF TECHNOLOGY
Lund University

Department of Information Technology

Computer Architecture, EDT030

Exam, Tuesday, March 7, 2000, 8.00 - 12.00 am

The exam consists of a number of assignments with a total of 100 points.

Grading: 40p ≤ grade 3 < 60p ≤ grade 4 < 80p ≤ grade 5

Instructions:

- *You may use a pocket calculator* in this exam but no other aids.
- Please *start each assignment on a new sheet*.
- *Write you name on each sheet of paper* that you hand in.
- Answers can be given in *Swedish* or *English*, however, *do not mix*.
- *You must motivate you answers thoroughly.* If there, in your opinion, is not enough information to solve an assignment, you can make reasonable assumptions that you need. *State these clearly.*
- You are free to use the information in *appendices A and B* for any of the assignments, *if needed*.

GOOD LUCK!

Assignment 1

a) One of the quantitative principles for computer design is the *CPU Performance Equation* which can be written: $T_{exe} = IC \times CPI \times T_c$

Define end *explain* all terms in the equation and *show* how it can be extended to take cache misses into account. (4 p)

b) Another quantitative principle is *Amdahl's law.* Formulate it as a formula and explain in your own words its consequences. (4 p)

c) Consider two processors, A and B, that have the same basic instruction set, but processor B has a new complex instruction that was found to be able to replace two simpler instructions without increasing the CPI. However, it lead to a lower clock frequency. Processor B can execute in 350 MHz while processor A could run in 400 MHz.

How large fraction of the code running on processor A must be replaced by the new instruction in order to reach the same performance for processor B with the lower clock frequency? (4 p)

d) In the assignent above, processor A does not have a floating point unit while processor B has one. This is also a contributing reason to the lower clock frequency of processor B. The floating point unit in processor B is fully pipelined and you can assume that the resulting CPI in average is 1.5. Floating point operations for programs in processor A are emulated by a sequence of integer instructions according to the table below. The integer CPI is 1.3 on average.

Floating point operation	**Number of integer instructions to emulate**
Load/Store/Move	1
Add/Sub/Compare/Other	10
Multiplication	12
Division	24

Assume that the program ***doduc*** from SPECfp92 executes 123 million instructions on processor A and that the new complex instruction according to assignment c) can be used instead of 12% of the original *integer instructions.*

Calculate the following for both processor A and B given the assumptions above:

- Execution time
- MIPS
- MFLOPS

(8 p)

Assignment 2

a) What are the design tradeoffs between a *large register file* and a *large data cache*? (4 p)

b) A sequence of instructions can exhibit *data dependencies* and two kinds of *name dependencies*. Define these dependencies and what kind of hazards they lead to if not taken care of properly in the processor implementation. (6 p)

c) What is *loop unrolling* and how can it affect cache performance, compiler design and pipeline performance? (5 p)

d) Suppose that it was possible to design a unified (common to instructions and data) first-level cache with two ports so that there would be no structural hazard in the the pipeline. What are the design trade-offs between using the unified cache compared to separate instruction and data cache? The total amount of cache memory should of course be the same in both cases. (5 p)

Assignment 3

Consider the following DLX assembly program (the destination operand is to the left):

```
0     add     r3,r31,r2
1     lw      r6,0(r3)
2     andi    r7,r5,#3
3     add     r1,r6,r0
4     srl     r7,r0,#8
5     or      r2,r4,r7
6     sub     r5,r3,r4
7     add     r15,r1,r10
8     lw      r6,0(r5)
9     sub     r2,r1,r6
10    andi    r3,r7,#15
```

Assume the use of a four-stage pipeline: fetch (IF), decode/issue (DI), execute (EX) and write back (WB). Assume that all pipeline stages take one clock cycle except for the execute stage. For simple integer arithmetic and logical instructions, the execute stage takes one cycle, but for a load from memory, five cycles are needed in the execute stage.

Suppose we have a simple scalar pipeline but allow some sort of out-of-order execution that results in the following table for the first seven instructions:

Instruction	IF	DI	EX	WB
`0: add r3,r31,r2`	0	1	2	3
`1: lw r6,0(r3)`	1	2	4[a]	9
`2: andi r7,r5,#3`	2	3	5[b]	6
`3: add r1,r6,r0`	3	4	10	11
`4: srl r7,r0,#8`	4	5	6	7
`5: or r2,r4,r7`	5	6	8	10
`6: sub r5,r3,r4`	6	7	9	12[c]

A number in the table indicates the clock cycle a certain instruction starts at a pipeline stage.

There are a lot of implementation details that can be deduced from the execution table above.

a) Explain why the first lw-instruction (instruction 1) cannot start in the execute stage until clock cycle 4. (3 p)

b) Explain why the first and-instruction (instruction 2) cannot start the execution stage until clock cycle 5. (3 p)

c) Explain why the first sub-instruction (instruction 6) cannot start the write back stage until clock cycle 12. (3 p)

d) Complete the table for the remaining instructions. (5 p)

e) Suppose instruction 2 was changed to: **`and r6,r5,#3`**. What implications would that have on the design of the pipeline? How would the table look like? (6 p)

Assignment 4

Assume a processor with a standard five-stage pipeline (IF, ID, EX, MEM, WB) and a branch prediction unit (a branch history table) in the ID-stage. Branch resolution is performed in the EX-stage. There are four cases for conditional branches:

- The branch is not taken and correctly predicted as not taken (NT/PNT)
- The branch is not taken and predicted as taken (NT/PT)
- The branch is taken and predicted as not taken (T/PNT)
- The branch is taken and correctly predicted as taken (T/PT)

Suppose that the branch penalties with this design are:

- NT/PNT: 0 cycles
- T/PT: 1 cycle
- NT/PT, T/PNT: 2 cycles

a) Describe how the CPU Performance Equation (see assignment 1 a) can be modified to take the performance of the branch prediction unit into account. Define the information you need to know to assess the performance. (5 p)

b) Use the answer in the assignment above to *calculate the average CPI* for the processor assuming a base CPI of 1.2. Assume 20% conditional branches (disregard from other branches) and that 65% of these are taken on average. Assume further that the branch prediction unit mispredicts 12% of the conditional branches. (5 p)

c) In order to increase the clock frequency from 500 MHz to 600 Mhz, a designer splits the IF-stage into two stages, IF1 and IF2. This makes it easier for the instruction cache to deliver instructions in time. This also affects the branch penalties for the branch prediction unit as follows:

- NT/PNT: 0 cycles
- T/PT: 2 cycles
- NT/PT, T/PNT: 3 cycles

How much faster is this new processor than the previous that runs on 500 MHz? (6 p)

d) Propose a solution to reduce the average branch penalty even further. (4 p)

Assignment 5

a) In a program for a parallel computer that supports a shared memory in hardware the programmer has declared two variables in the following way:

```
int data_updated_by_thread1;
char dummy[28];
int data_updated_by_thread2;
```

The two variables are accessed (both read and written) by two different threads (or processes) but not shared.

Give a *plausible explanation* to why the programmer has chosen to put a dummy variable of 28 bytes between the two integer variables. (4 p)

b) Suppose you have written a program with a loop that executes 500 iterations per second on an ordinary sequential computer. The loop is parallel and you want to improve performance by executing it on a multiprocessor with four processors. Even though the actual computation in the loop is perfectly parallelisable, there will be communication that on the parallel computer you intend to use takes 750 µs for each iteration.

What is the speedup of the parallel program compared to the original sequential code? (4 p)

c) In parallel computers with physically shared memory a *split-transaction bus* (sometimes also called a pipelined or packet-switched bus) is often used for the processor-memory bus.

Explain the characteristics of a split-transaction bus that distinguishes it from ordinary synchronous processor-memory buses. (4 p)

d) Why are split-transaction buses especially useful in bus-based multiprocessors with shared memory? (4 p)

e) What distinguishes directory-based cache coherence schemes from snoopy-based cache coherence schemes in multiprocessor with shared memory? Why are they useful? (4 p)

Appendix A: DLX Standard Instruction Set

Instruction type/opcode	Instruction meaning
Data transfers	Move data between registers and memory, or between the integer and FP or special registers; only memory address mode is 16-bit displacement + contents of a GPR
LB,LBU,SB	Load byte, load byte unsigned, store byte
LH,LHU,SH	Load half word, load half word unsigned, store half word
LW,SW	Load word, store word (to/from integer registers)
LF,LD,SF,SD	Load SP float, load DP float, store SP float, store DP float
MOVI2S,MOVS2I	Move from/to GPR to/from special register
MOVF,MOVD	Copy one FP register or a DP pair to another register or pair
MOVFP2I,MOVI2FP	Move 32 bits from/to FP registers to/from integer registers
Arithmetic/logical	Operations on integer or logical data in GPRs; signed arithmetic trap on overflow
ADD,ADDI,ADDU,ADDUI	Add, add immediate (all immediates are 16 bits); signed and unsigned
SUB,SUBI,SUBU,SUBUI	Subtract, subtract immediate; signed and unsigned
MULT,MULTU,DIV,DIVU	Multiply and divide, signed and unsigned; operands must be FP registers; all operations take and yield 32-bit values
AND,ANDI	And, and immediate
OR,ORI,XOR,XORI	Or, or immediate, exclusive or, exclusive or immediate
LHI	Load high immediate—loads upper half of register with immediate
SLL,SRL,SRA,SLLI SRLI,SRAI	Shifts; both immediate (S__I) and variable form (S__); shifts are shift left logical, right logical, right arithmetic
S__, S__I	Set conditional: "__" may be LT, GT, LE, GE, EQ, NE
Control	Conditional branches and jumps; PC-relative or through register
BEQZ, BNEZ	Branch GPR equal/not equal to zero; 16-bit offset from PC+4
BFPT, BFPF	Test comparison bit in the FP status register and branch; 16-bit offset from PC+4
J, JR	Jumps; 26-bit offset from PC+4 (J) or target in register (JR)
JAL, JALR	Jump and link; save PC+4 in R31, target is PC-relative (JAL) or a register (JALR)
TRAP	Transfer to operating system at a vectored address
RFE	Return to user ocde from an exception; restore user mode
Floating point	FP operations on DP and SP formats
ADDD,ADDF	Add DP, SP numbers
SUBD,SUBF	Subtract DP, SP numbers
MULTD, MULTF	Multiply DP, SP floating point
DIVD, DIVF	Divide DP, SP floating point
CVTF2D, CVTF2I, CVTD2F, CVTD2I, CVTI2F, CVTI2D	Convert instructions: CVTx2y converts from type x to y, where x and y are I (integer), D (double precision), or F (single precision). Both operands are FPRs.
__D, __F	DP and SP compares: "__" = LT, GT, LE, GE, EQ, NE; sets bits in FP status register

Appendix B: Instruction mix measurements

DLX instruction mix for five programs from SPECint92:

Instruction	compress	eqntott	espresso	gcc(cc1)	li	Integer average
load	19.8%	30.6%	20.9%	22.8%	31.3%	26.0%
store	5.6%	0.6%	5.1%	14.3%	16.7%	9.0%
add	14.4%	8.5%	23.8%	14.6%	11.1%	14.0%
sub	1.8%	0.3%		0.5%		0.0%
mul				0.1%		0.0%
div						0.0%
compare	15.4%	26.5%	8.3%	12.4%	5.4%	13.0%
load imm	8.1%	1.5%	1.3%	6.8%	2.4%	3.0%
cond. Branch	17.4%	24.0%	15.0%	11.5%	14.6%	16.0%
uncond branch	1.5%	0.9%	0.5%	1.3%	1.8%	1.0%
call	0.1%	0.5%	0.4%	1.1%	3.1%	1.0%
return, jmp ind	0.1%	0.5%	0.5%	1.5%	3.5%	1.0%
shift	6.5%	0.3%	7.0%	6.2%	0.7%	4.0%
and	2.1%	0.1%	9.4%	1.6%	2.1%	3.0%
or	6.0%	5.5%	4.8%	4.2%	6.2%	5.0%
other (xor, not)	1.0%		2.0%	0.5%	0.1%	1.0%
other FP						0.0%

DLX instruction mix for five programs from SPECfp92:

Instruction	doduc	car	hydro2d	mdijdp2	su2cor	FP average
load	1.4%	0.2%	0.1%	1.1%	3.6%	1.0%
store	1.3%	0.1%		0.1%	1.3%	1.0%
add	13.6%	13.6%	10.9%	4.7%	9.7%	11.0%
sub	0.3%		0.2%		0.7%	0.0%
mul						0.0%
div						0.0%
compare	3.2%	3.1%	1.2%	0.3%	1.3%	2.0%
load imm	2.2%		0.2%	2.2%	0.9%	1.0%
cond. branch	8.0%	10.1%	11.7%	9.3%	2.6%	8.0%
uncond branch	0.9%	0.4%		0.4%	0.1%	0.0%
call	0.5%	1.9%			0.3%	1.0%
return, jmp ind	0.6%	1.9%			0.3%	1.0%
shift	2.0%	0.2%	2.4%	1.3%	2.3%	2.0%
and	0.4%	0.1%			0.3%	0.0%
or		0.2%	0.1%	0.1%	0.1%	0.0%
other (xor, not)						0.0%
load FP	23.3%	19.8%	24.1%	25.9%	21.6%	23.0%
store FP	5.7%	11.4%	9.9%	10.0%	9.8%	9.0%
add FP	8.8%	7.3%	3.6%	8.5%	12.4%	8.0%
sub FP	3.8%	3.2%	7.9%	10.4%	5.9%	6.0%
mul FP	12.0%	9.6%	9.4%	13.9%	21.6%	13.0%
div FP	2.3%		1.6%	0.9%	0.7%	1.0%
compare FP	4.2%	6.4%	10.4%	9.3%	0.8%	6.0%
mov reg-reg FP	2.1%	1.8%	5.2%	0.9%	1.9%	2.0%
other FP	2.4%	8.4%	0.2%	0.2%	1.2%	2.0%

Short answers and/or solutions to final exam in ComputerArchitecture 2000-03-07

Assignment 1

a) The terms are defined as follows:

- *IC* – Instruction count of a given program (or average of a set of programs)
- *CPI* – The average number of clock cycles per instruction for a given program (or a set of programs)
- T_c – The clock cycle time for an implementation

To take cache misses into account we need to extend the *CPI*. Call the original *CPI* for CPI_{base}. The new *CPI* is then: $CPI_{new} = CPI_{base}$ + "Memory references per instruction" · "Miss rate" · "Miss penalty"

b) Amdahl's law: $Speedup_{overall} = \dfrac{1}{(1 - Fraction_{enhanced}) + \dfrac{Fraction_{enhanced}}{Speedup_{enhanced}}}$

$Fraction_{enhanced}$ is the fraction of the execution time a particular enhancement to the system can be used. $Speedup_{enhanced}$ is the factor the enhancement speeds up the execution when it can be used.

The meaning of Amdahl's law is that if an enhancement cannot be used for a substantial fraction of the time, then its impact on the overall performance is limited. Thus: *make the common case fast*!

c) This is a typical application of Amdahl's law and the CPU Performance Equation combined. Let's start with the performance equation. For the two processors to be equally fast, in spite of different clock frequencies, the number of instructions executed by processor B must be less than the what processor A executes.

$IC_A \cdot CPI \cdot 1/400 = IC_B \cdot CPI \cdot 1/350, \Rightarrow IC_A \approx IC_B \cdot 1.1429$

We also know that the instruction set is the same, it is only a fraction, α, of the original instructions, IC_A, that can be replaced with half as many instructions. Thus:

$IC_B = \alpha \cdot IC_A / 2 + (1-\alpha)\, IC_A = (\alpha/2 + (1-\alpha)) \cdot IC_A$

If this is combined with the result above we get: $\alpha \approx 0.25$

d) The number of instructions (both integer and floating point) on processor A is 123 million. This includes floating point operations emulated with integer instructions. The fraction of floating point operations in appendix B refers to a processor that has a floating point unit, such as processor B. The fraction of integer instructions is: 34.4%. Calculating the instruction count of processor B, IC_B', we thus get:

$123 = IC_B' \cdot (0.344 + 1\cdot(0.233+0.057+0.021) + 10\cdot(0.088+0.038+0.042+0.024) + 12\cdot 0.12 + 24\cdot 0.023) = IC_B'\cdot 4.567 \Rightarrow IC_B' = 26.93$ million instructions.

However, this is without taking into account that 12% of the integer instructions on processor B could be replaced by half as many new complex instructions. This means that the real instruction count for processor B is: $IC_B = IC_B' \cdot (0.344\cdot(0.12/2 + 0.88) + (1-0.344))$. Entering 26.93 for IC_B' we get $IC_B = 26.37$ million instructions.

The number of (unnormalised) floating point operations is: $26.93\cdot(1-0.344) \cdot 10^6 = 17.67$ million floating operations.

Observe! The instruction count IC_B' is used here since the statistics in appendix B is for the unmodified instruction set.

We are now ready to calculate the values we need. First for processor A:

- $T_{exeA} = IC_A \cdot CPI_A \cdot T_{cA} = 123 \cdot 1.3 \cdot 1/400 = 0.40$ s.
- $MIPS_A = IC_A/T_{exeA} = 123 / 0.4 = 307.5$ million instructions per second
- $MFLOPS_A = 17.67 / 0.4 = 44.18$ million floating point operations per second

For processor B:

- $T_{exeB} = IC_B \cdot CPI_B \cdot T_{cB} = 26.37 \cdot 1.3 \cdot 1/350 = 0.098$ s.
- $MIPS_B = IC_B/T_{exeB} = 26.37 / 0.098 = 269.08$ million instructions per second

- $MFLOPS_B = 17.67 / 0.098 = 180.31$ million floating point operations per second

The moral of the story is that using the MIPS and MFLOPS numbers as performance metrics can give contradictory results. Processor A has a higher MIPS number than processor B but a much lower MFLOPS number for this particular program.

Assignment 2

a) A large register file and a large data cache both serve the purpose of reducing memory traffic. From an implementation point of view, the same chip area can be used for either a large register file or a large data cache. With a larger register set, the instruction set must be changed so that it can address more register in the instructions. From a programming point of view, registers can be manipulated by program code, but the data cache is transparant to the user. In fact, the data cache is primarily involved in load/store operations. The addressing of a cache involves address translation and is more complicated than that of a register file.

b) A *data dependence* occurs when an instruction has a source operand that is the destination operand of a previous instruction.

There are two kinds of name dependencies:

- *output dependence* – two instructions use the same destination operand
- *antidependence* – an instruction has a destination operand used as a source operand in a previous instruction

These dependencies all prohibit instruction reordering. If a data dependence is not taken care of in the pipeline implementation we will have a read-after-write, RAW, hazard. If an output dependence is not taken care of there is a write-after-write, WAW, hazard and finally, if an antidependence is not taken care of, there is a write-after-read, WAR, hazard.

c) In loop unrolling the loop body is repeated a number of times to increase the basic block length and to reduce the number of branch instructions. The pipeline performance is likely improved. In a processor that allows out-of-order execution with code rescheduling in hardware there are more instructions to execute in parallel. If the compiler support code rescheduling it can also move instructions so that the number of stall cycles are reduced. It requires that the compiler designer knows about this particular pipeline implementation in order to perform a correct and useful instruction rescheduling.

It probably reduces the performance of the cache memory since instructions are not reused to the same degree as it was before. The temporal locality is reduced.

d) With a unified cache, the fraction of the cache devoted to instructions and data respectively may change from one program to another. This can be a benefit but also a problem for, e.g., a small loop that touches a lot of data. In a unified cache, instructions will probably be replaced by data as the loop executes. With two separate caches, the entire loop can fit in cache and performance will probably be improved.

The design trade-offs involve choosing the correct division between instruction and data caches for separate cache memories, studying the miss rate for a unified cache, choosing a correct address mapping for a unified cache and a replacement policy.

Assignment 3

a) Because there is no forwarding from the WB-stage and the correct value of source register r3 is therefore not available until clock cycle 4.

b) Only one instruction can be issued in each clock cycle and since instruction 1 has to wait, instruction 2 also must wait one clock cycle.

c) The write back stage is occupied in clock cycles 10 and 11 by instructions that have been issued earlier.

d) Here is the continuation of the table:

Instruction	IF	DI	EX	WB	Comment
`6: sub r5,r3,r4`	6	7	9	12	

Instruction	IF	DI	EX	WB	Comment
`7: add r15,r1,r10`	7	8	12	13	Wait for register r1
`8: lw r6,0(r5)`	8	9	13	18	Wait for register r5
`9: sub r2,r1,r6`	9	10	19	20	Wait for register r6
`10: andi r3,r7,#15`	10	11	14	15	Can execute out-of-order

e) The main implication is that instruction 2 is not allowed to execute out-of-order in relation to instruction 1. There is a name-dependence between these instructions and if instruction 2 completes before instruction 1 there will be a WAW-hazard. Instruction 2 is stalled in the DI stage and the table must be modified:

Instruction	IF	DI	EX	WB
`0: add r3,r31,r2`	0	1	2	3
`1: lw r6,0(r3)`	1	2	4	9
`2: andi r6,r5,#3`	2	3	**9**	**10**
`3: add r1,r6,r0`	3	4	11	12
`4: srl r7,r0,#8`	4	5	6	7
`5: or r2,r4,r7`	5	6	8	11
`6: sub r5,r3,r4`	6	7	9	13

Assignment 4

a) The instruction count and the clock cycle time are not affected. Therefore the only modification is in the CPI count which will have an addition that comes from branch instructions. The addition depends on the relative frequency of conditional branch instructions, f_{branch}, the fraction of these that are taken, b_{taken}, and the fraction of the branches that are miss-predicted, $b_{misspred}$.

$CPI = CPI_{base} + f_{branch} \cdot (b_{taken} \cdot (b_{misspred} \cdot 2 + (1-b_{misspred}) \cdot 1) + (1-b_{taken}) \cdot (b_{misspred} \cdot 2 + (1-b_{misspred}) \cdot 0))$

b) $f_{branch} = 0.2$, $b_{taken} = 0.65$, $b_{misspred} = 0.12$.

$CPI = 1.2 + 0.2 \cdot (0.65 \cdot (0.12 \cdot 2 + 0.88 \cdot 1) + 0.35 \cdot (0.12 \cdot 2 + 0.88 \cdot 0)) = 1.2 + 0.2 \cdot (0.728 + 0.084) = 1.3624$

c) $CPI_{new} = 1.2 + 0.2 \cdot (0.65 \cdot (0.12 \cdot 3 + 0.88 \cdot 2) + 0.35 \cdot (0.12 \cdot 3 + 0.88 \cdot 0)) = 1.2 + 0.2 \cdot (1.378 + 0.126) = 1.5008$

$T_{exeold} = IC \cdot CPI_{old} \cdot 1/500$

$T_{exenew} = IC \cdot CPI_{new} \cdot 1/600$

$Speedup = T_{exeold} / T_{exenew} = (CPI_{old} \cdot 1/500) / (CPI_{new} \cdot 1/600) = (1.3624 / 500) / (1.5008 / 600) = 1.089$

The new processor with higher clock frequency is thus only 8.9% faster than the old even though the clock frequency has been increased by 20%. The sole reason for this is the branch hazards.

d) A *branch target buffer* can be used in the IF-stage. The branch penalty for correctly predicted branches will then be 0.

Assignment 5

a) The block size of the multiprocessor is probably 32 bytes. The programmer do not want the two variables to be placed in the same cache block, as this will lead to *false sharing*, and ensures this by putting them 28 bytes apart in the memory.

b) In the original program, each loop iteration takes 1/500 s = 0.002 s

The the parallel program, the loop is parallelised over four processors executing four iterations in parallel. One iterations takes on average 0.002/4 = 0.0005 s. The communication takes 750 μs for each iteration. The total amount of time per iteration is: 0.00125. The speedup is 0.002/0.00125 = 1.6.

c) In a split-transaction bus, the request operation, of, e.g. a read operation, is separated from the reply. If a processor reads from a memory, it relinquishes the bus when the request has been made. The memory module has to arbitrate for the bus and then sends the reply with the memory contents back to the processor. Between the request and the reply, the bus is free to be used by other bus masters.

d) Split-transaction buses are particularly useful in multiprocessors since each processor is a potential bus master and the bus can be better utilised if it is not blocked when one processor is performing a memory operation.

e) Snoopy cache coherence protocols rely upon the broadcast nature of a shared bus. For scalable multiprocessors a shared bus cannot be used since it would be choked by the amount of communication. Therefore, scalable interconnection networks are used. These do not support broadcast operations well and directory based cache coherence protocols are therefore used. In such a protocol, a directory is kept at the (distributed) memory and this directory keeps track of the locations of possible copies of a memory block that reside in the caches of different processors. When a coherence operation needs to be performed, the directory is consulted and separate messages are sent (by the hardware) to the processor that needs it.