

CS8803 Languages & Compilers for Embedded System
Summer 2009
Homework Assignment #1
Assigned 05/25/09 Due in the **first 5 min** in class 06/01/09

1. (10%) Please do research and elaborate the differences between Intel's hyper-threading (HT Technology) in Pentium 4 Processor and the SMT (Simultaneous Multithreading) supported in the Compaq Alpha 21464 (EV8).

Intel's hyper-threading in Pentium 4	SMT in Compact Alpha 21464
<p>2-way SMT</p> <p>Replicated hardware contexts</p> <ul style="list-style-type: none">- Next instruction pointer- Instruction stream buffers- ITLB- Return stack predictor- APIC- Register Alias Tables <p>Shared resources</p> <ul style="list-style-type: none">- Rename register pool- Caches- TLB	<p>4-way SMT</p> <p>Replicated hardware contexts</p> <ul style="list-style-type: none">- Program counter- Architected registers <p>Shared resources</p> <ul style="list-style-type: none">- Rename register pool- Instruction queue- Caches- TLB- Branch predictors

2. (30%) Consider the following assembly code and answer the subsequent questions.

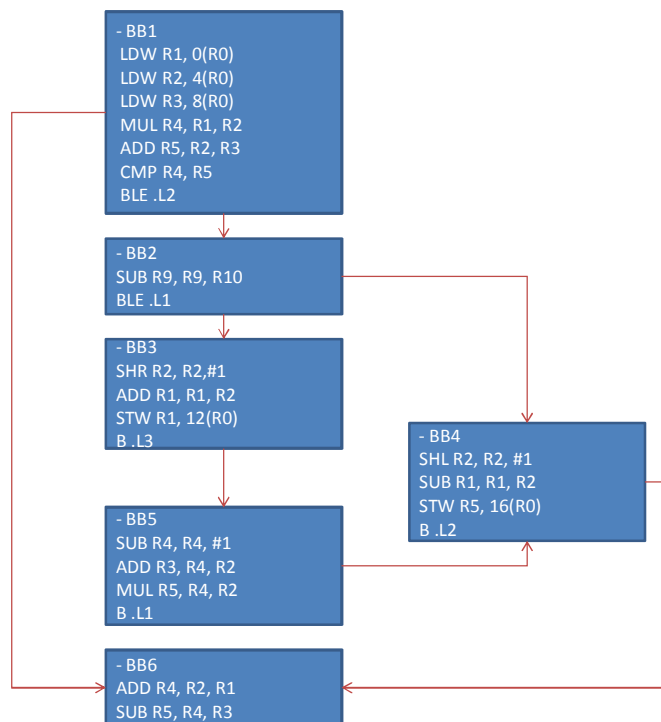
```
Main:
    LDW R1, 0(R0)
    LDW R2, 4(R0)
    LDW R3, 8(R0)
    MUL R4, R1, R2
    ADD R5, R2, R3
    CMP R4, R5
    BLE .L2
    SUB R9, R9, R10
    BLE .L1
    SHR R2, R2, #1
    ADD R1, R1, R2
    STW R1, 12(R0)
    B .L3

.L1:
    SHL R2, R2, #1
    SUB R1, R1, R2
    STW R5, 16(R0)
    B .L2

.L3:
    SUB R4, R4, #1
    ADD R3, R4, R2
    MUL R5, R4, R2
    B .L1

.L2:
    ADD R4, R2, R1
    SUB R5, R4, R3
```

2.1. (10%) Draw the control flow graph (CFG) for the above code.



2.2. (10%) Assume that we have a perfect branch predictor and conditional branches are always NOT taken in the code. Calculate the maximum ILP based on a trace scheduler. Do not rename registers.

Trace: BB1 -> BB2 -> BB3-> BB5-> BB4-> BB6

1	LDW R1, 0(R0)	LDW R2, 4(R0)	LDW R3, 8(R0)	SUB R9, R9, R10
2	MUL R4, R1, R2	ADD R5, R2, R3		
3	CMP R4, R5	SHR R2, R2, #1		
4	BLE .L2	BLE .L1	ADD R1, R1, R2	SUB R4, R4, #1
5	STW R1, 12(R0)	ADD R3, R4, R2	MUL R5, R4, R2	B .L3
6	SHL R2, R2, #1	STW R5, 16(R0)	B .L1	
7	SUB R1, R1, R2	B .L2		
8	ADD R4, R2, R1			
9	SUB R5, R4, R3			

ILP = 23/9 = 2.5556

2.3. (10%) Continuing from Problem 2.2, now we have a more realistic machine call GT-Force which consists of two execution pipes: the U-pipe and V-pipe, similar naming scheme to the original Intel p54c Pentium processor. The U-pipe is a primary pipeline in which all instructions can be executed while the V-pipe is a secondary pipeline wherein only ADD, SUB, and branch instructions are allowed. In addition, Write-After-Read (WAR) semantics in an instruction pair is allowed between U and V. In other words, a destination read by U-pipe will always precede the write to the same destination by V-pipe but not the other way around. Instructions issued to both pipes in the same cycle must satisfy these conditions. Given all instructions have a unit latency, please show the best possible IPC you could get. (Use the assumption of prob 2.2. Do not rename registers. You also need to consider dependences.)

Cycle	U-PIPE	V-PIPE
1	LDW R1, 0(R0)	SUB R9, R9, R10
2	LDW R2, 4(R0)	
3	LDW R3, 8(R0)	
4	MUL R4, R1, R2	ADD R5, R2, R3
5	CMP R4, R5	SUB R4, R4, #1
6	SHR R2, R2, #1	BLE .L2
7	ADD R1, R1, R2	BLE .L1
8	STW R1, 12(R0)	ADD R3, R4, R2
9	MUL R5, R4, R2	B .L3
10	SHL R2, R2, #1	B .L1
11	STW R5, 16(R0)	SUB R1,R1, R2
12	B .L2	ADD R4, R2, R1
13	SUB R5, R4, R3	

IPC = 23 / 13 = 1.769

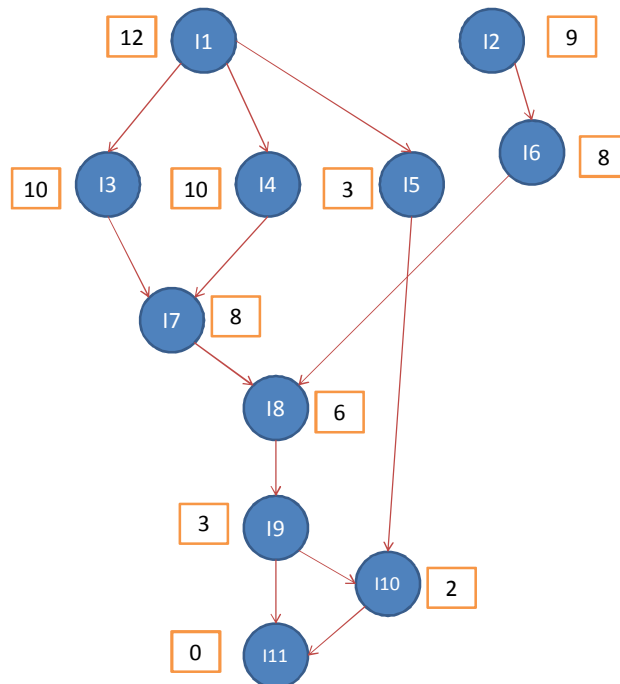
3. (20%) Consider the following code. Each array expression indicates a memory instruction. (e.g., $A = X[0]$ will load the value A from memory location $X[0]$.) Please answer the following questions.

```
I1: A = X[0]
I2: B = Y[0]
I3: C = A + 1
I4: D = A * 2
I5: E = A + 3
I6: F = B + 4
I7: G = C * D
I8: G = G * F
I9: H = Y[G]
I10: I = E + H
I11: J = H * I
```

Functional units and latencies of each FU

- Two integer add and subtraction unit: 1 cycle
- One integer multiplication unit: 2 cycles
- One load and store unit: 3 cycles

3.1. (10%) Draw the data dependence graph for the above code. (You can simply write down in instruction id such as i8 for each node.)



3.2. (10%) Schedule the instructions using list scheduling and also draw and fill the reservation table with the scheduled instructions. (Like the table on page 19 in the Lec3 slide.)

Schedule = {I1, {I3, I4}, I2, {I6, I7}, I8, {I5, I9}, I10, I11}

- Reservation Table

Time	MEM	ADDER1	ADDER2	MULT
0	I1			
1				
2				
3	I2	I3		I4
4				
5				I7
6		I6		
7		I5		I8
8				
9	I9			
10				
11				
12		I10		
13				I11

4. (10%) The following diagram shows that the state of the expand instruction buffers after fetching instructions from the instruction cache. Based on the diagram, draw the figure of the instruction cache line when using following encoding schemes:

IBUF0	W00	W01		W03	W04		W06	
IBUF1		W09	W10	W11		W13		

(1) VLIW Fixed-Overhead Encoding

11011010	W00	W01	W03	W04	W06	01110100	W09	W10	W11	W13
----------	-----	-----	-----	-----	-----	----------	-----	-----	-----	-----

(2) VLIW Distributed Encoding

1	W00	1	W01	1	W03	1	W04	0	W06	1	W09	1	W10	1	W11	0	W13
---	-----	---	-----	---	-----	---	-----	---	-----	---	-----	---	-----	---	-----	---	-----

(3) Template-based VLIW Encoding. (Assume W00 is the beginning of a new template and each template. Use the simple form discussed in slide 18 of Lecture 4)

Temp0	W00	W01	W03	Temp1	W04	W06	W09	Temp2	W10	W11	W13
-------	-----	-----	-----	-------	-----	-----	-----	-------	-----	-----	-----

Temp0 = (0->0), (1->1), (2->3), chain

Temp1 = (0->4), (1->6); seq, (2->1),chain

Temp2 = (0->2), (1->3), (2->5); seq

5. (10%) Given two VLIW machine implementations called KU3 and KU7. KU3 can dispatch and execute 3 operations (or syllables) per cycle while KU7 can do 7. Assume all functional units are completely replicated for both machines, i.e., there is no resource hazard. All operations take one cycle to execute and there is no cache misses or branch-related penalty. Given the VLIW code sequence with stop bits (shown as a bold vertical bar: |) below, how many execution cycles are needed for KU3 and KU7? (Note that, M, I, B, F are distinct function units.)

VLIW I0	M	I	I
VLIW I1	M	 M	I
VLIW I2	M	M	 I
VLIW I3	M	I	B
VLIW I4	M	F	I
VLIW I5	M	 M	I
VLIW I6	M	I	 I
VLIW I7	M	F	B
VLIW I8	M	I	 I
VLIW I9	B	B	 B

KU3		
M	I	I
M		
M	I	M
M	I	
M	I	B
M	F	I
M		
M	I	M
I		
I	M	F
B	M	I
I	B	B
B		

KU7						
M	I	I	M			
M	I	M	M	I		
M	I	B	M	F	I	M
M	I	M	I			
I	M	F	B	M	I	
I	B	B	B			

KU3: 13 Cycles

KU7: 6 Cycles

6. (20%) The GT_RISC is a reduced version of its counter part GT_CISC. Consider a program P with the following profiles of static and dynamic instructions based on GT_CISC. The table also shows that how the GT_RISC instructions are translated from GT_CISC instructions.

GT_CISC ISA (length)	# of static CISC insns (latency)	# of dynamic CISC insns	GT_RISC ISA
ADD (4B)	41 (1)	1480	ADD
MADD (6B)	64 (3)	458	ADD + MUL
BEQ (9B)	15 (1)	281	CMP + BR
MEMADD (14B)	43 (4)	142	LD + ADD + STORE

6.1 (10%) The GT_CISC instruction has variable length as specified in the first column while the GT_RISC has a uniform size: 4 bytes (4B). Please evaluate the code sizes of these two ISAs for this program. (Assume you do not encounter any register fill and spill)

ISA	CODE SIZE
GT_CISC	4 bytes * 41 + 6 bytes * 64 + 9 bytes * 15 + 14 bytes * 43 = 1285 bytes
GT_RISC	4 bytes * (1 * 41 + 2 * 64 + 2 * 15 + 3 * 43) = 1312 bytes

6.2 (10%) The instruction latencies for GT_CISC are shown in the 2nd column in the table while the GT_RISC are all 1 cycle. Which one gives you better “performance per byte”? (Assume instruction cache is perfect.)

Define the performance as 1/cycles.

Cycles in GT_CISC: $1480 * 1 + 458 * 3 + 281 * 1 + 142 * 4 = 3703$ cycles

Cycles in GT_RISC: $1480 * 1 + 458 * 2 + 281 * 2 + 142 * 3 = 3384$ cycles.

Performance per byte in GT_CISC: $1/(3703 \text{ cycles} * 1285 \text{ bytes}) = 2.1016 * 10^{-7} /(\text{cycles} * \text{bytes})$

Performance per byte in GT_RISC: $1/(3384 \text{ cycles} * 1312 \text{ bytes}) = 2.2523 * 10^{-7} /(\text{cycles} * \text{bytes})$

Thus, GT_RISC gives the better “performance per byte” than GT_CISC.