

CS8803 Languages & Compilers for Embedded System

Summer 2009

Homework Assignment #2

Assigned 06/17/09

Due in the **first 5 min** in class 06/29/09

1. (15%) Please do research and compare the distinguished architectural and design features for the following three commercial processors: ARM's Cortex A9, VIA's Nano (Isaiah), and Intel's Atom. Also, please discuss their design philosophies and present your view about which type of applications would perform best on each respective architecture.
2. (15%) In code compression schemes, the compressed instructions are stored at completely different memory locations from the uncompressed instructions, causing a problem when fetching instructions from I-cache misses. Please read the papers titled "Executing Compressed Programs on An Embedded RISC Architecture" by Wolfe and Chanin (obtain it from the weblink in the course web) and "CodePack: Code Compression for PowerPC Processors" and explain how they address the problem in detail.
3. (15%) Most of the current processors have SIMD (Single Instruction Multiple Data) extensions such as IBM's AltiVec, Intel's MMX/SSE, and AMD's 3DNow!. Although they were defined to achieve similar goals in exploiting data parallelism, they do contain different features. Please do research and discuss the differences between IBM's AltiVec and Intel's MMX/SSE.

4. (40%) Assume a 32-bit architecture which has SIMD extensions. There are 16 128-bit SIMD single precision floating point registers (Q0-Q15) and all the arithmetic instructions perform 4-way packed SIMD operations. Also, a number of instructions are supported for SIMD operations. For example, `LDDQ` loads 128-bit data into a vector register, and `LDHQ` or `LDLQ` load 64-bit data into the higher or lower 64-bit of SIMD registers and leave the other half intact. Also, `SHUF` moves data from the source registers to the destination register using the immediate parameter to determine where the data comes from in the sources.

SHUF INSTRUCTION EXAMPLE:

```

Q1: X1 Y1 Z1 W1
Q2: X2 Y2 Z2 W2

SHUF Q3, Q1, Q2, 0xF8; Q3 = X1 X1 Y2 W2 (0xF8 = 11 11 10 00)
SHUF Q4, Q1, Q2, 0x36; Q4 = W1 X1 Z2 Y2 (0x36 = 00 11 01 10)
    
```

Instruction	Description
<code>LDW t, [addr]</code>	Load word
<code>LDH t, [addr]</code>	Load half word
<code>LDB t, [addr]</code>	Load byte
<code>LDDQ t, [addr]</code>	Load double quad
<code>LDHQ t, [addr]</code>	Load high quad
<code>LDLQ t, [addr]</code>	Load low quad
<code>STW [addr], s1</code>	Store word
<code>STH [addr], s1</code>	Store half word
<code>STB [addr], s1</code>	Store byte
<code>STDQ [addr], s1</code>	Store double quad
<code>STHQ [addr], s1</code>	Store high quad
<code>STLQ [addr], s1</code>	Store low quad
<code>VADD t, s1, s2</code>	Vector addition
<code>VSUB t, s1, s2</code>	Vector subtraction
<code>VMUL t, s1, s2</code>	Vector multiplication
<code>SHUF t, s1, s2, imm</code>	Shuffle instruction

4.1. (20%) Consider a data structure given below. Using the above instructions, write an assembly code which performs “swizzle” operation to enable “Structure-of-Array (SOA)” type of processing and performs vector addition. Assume that the structure data was declared and started at the memory location of 0x8000. The final swizzled packed data should be located in register Q11 to Q14, and the result of the addition should be placed into Q15. Try to minimize the number of instructions used in the code.

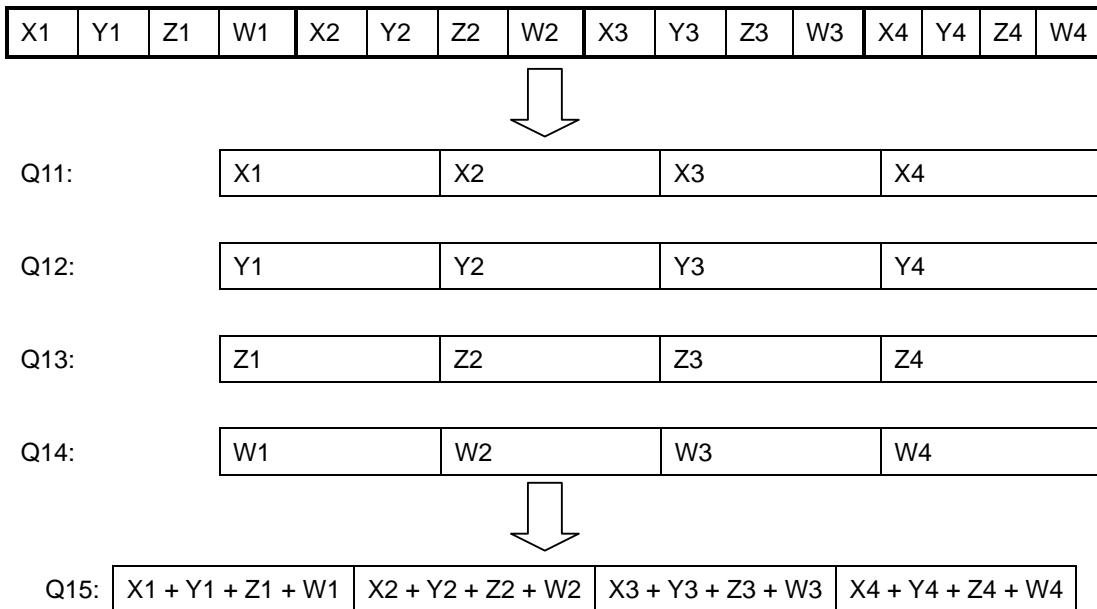
```

type struct {
    float x;
    float y;
    float z;
    float w;
} Vertex_t;

Vertex_t AOS[4];
/* 0x8000 is the start address of the AOS structure. */

```

Memory: 0x8000



- 4.2. (20%) Now assume that the architecture has new instructions of MOVHL and MOVLH which are able to manipulate SIMD registers. For example, "MOVHL Q2, Q1" moves lower half data of the register Q1 into upper half of the register Q2. Using these new instructions along with the instructions listed above, write an assembly code to perform the same horizontal addition same as the prob 4.1 but without the swizzle operation. Analyze and compare the number of instructions with the code you did in the prob. 4.1.
5. (15%) Consider a real-time system containing the following three tasks. $T_1 = (6, 2, 6)$, $T_2 = (4, 1, 4)$, $T_3 = (3, 0.5, 3)$. Assume that the tasks are independent and periodic. Please find a feasible clock-driven schedule based on the method discussed in class.

$$T_i = (p_i, e_i, D_i)$$

T_i : task

p_i : time period T_i repeated

e_i : worst case execution time of T_i

D_i : deadline within the period from released