Andrew Johnson

CS271 W15

Final Project

ARMv7 was created to be targeted at high-performance application processors. Processors used to OS such as Linux, Android, and Windows phones. So it ranges more near consumer electronics. These electronics, mobile phones, are most likely are powered by a Cortex applications processor. You can actually see these facts in the user manual of your phone. Sadly, no one actually reads those things, so most people are unaware. As it are highly used in mobile phones, it also has a significant use in network switches, gaming platforms, and other mainstream electronic devices.

ARMv7 is just upgraded more beefy version the later ARMv6, and likely uses a Cortex-A series core. The base architecture of it includes all ARM and Thumb instruction sets to help with code density along with performance. As well as 7 operating modes, which there is one lone operator (User Mode) which is not privileged. This mode is used to execute the program in its desired environment. ARMv7 also has Virtual memory, NEON technology, and a full debug and trace support. That last one seems to be a necessity for any programmer. Along with all the base features, it has options to include multiprocessing, have up to a 40 bit register, virtualization, floating point capabilities, and performance monitors. The couple main ones to note are 40 registers and floating point usage.

When first looking at the difference between ARM and IA-32, one of the first things to note are the data sizes used in each architecture. A byte is the same for both, but a word in ARM is twice the size as it would be in IA-32, being 32 bits instead of 16 bits. For ARM a halfword satisfies the 16 bit size. Then a quadword just gets its name changed to a doubleword. Essentially ARM cuts all of AI-32's data sizes in half. One good thing is that both architectures are capable of using floating point values. ARM uses 43 general-purpose registers, and 16 are usable at any one time. These registers are named like r0, r1, r2 up to 12 with 4 other registers.

All instructions are 32-bit for ARM instruction set, and for the Thumb instruction set, it uses 16-bit but Thumb-2 technology with Cortex processors adds 32-bit instructions to increase performance. Instead of having most of these registers have a specific purpose, ARM just has them be all 16 identical registers, although 3 of them do have a specific purpose while still being identical to the others.

With ARM having a RISC architecture, it makes it a load-store architecture. Which means the processing instructions can't directly affect the memory of the registers, it can only operate on the register itself. This is different from what we are used to, where IA-32 processing instructions can go straight in and operate on both memory and register. When using IA-32 we normally address I/O address space directly, but ARM doesn't have an equivalent to this and thinks all memory mapped is in a 4GB address space. Since all addresses are a fixed size you won't be able to encode a 32-bit address, so the address is held in a register that the instruction has specified. So you basically you are indirectly accessing memory through a GPR. This makes needing to load constants a lot meaning you must place them in memory and access them using LDR instruction.

When using ARM data processing it usually takes 3 operands, which is different to how we normally use just 2 operands in IA-32. For example, using the "add" instruction in IA-32 you would see something like,

```
mov eax, 5 ; eax = 5

mov ebx, 6 ;ebx = 6

add eax, ebx ; eax = eax + ebx
```

Although with ARM, when using the “add” instruction it’s as simple as loading a couple of values into r1 and r2, then use another register such as r0 to hold the added value. After you have loaded the values it’ll look something like this,

Add r0, r1, r2 which is equivalent to r0 = r1 + r2.

One thing to keep in mind is that a flag for dividing by zero, known as the Unidentified instruction exception, is actually an optional flag and so it doesn’t come with all ARM processors.

When using IA-32 we get the luxury of CALL and RET instructions, which place the return address on the stack and takes it off. ARM doesn’t have these luxuries. All it does is simply add and subtract a fixed amount from the stack pointer. It’s not entirely bad though because this allows leaf functions to avoid the stack altogether and doesn’t restrict the composition of the stack frame.

While using the stack frame, IA-32 uses a single instruction to push/pop GPR. ARM’s LDM/STM instructions are used similarly in a single operation. Although the ARM version is quite more useful such that any subset of the register can be pushed/popped in a single instruction. For example, PUSH would STR and POP would be LDR. When using the memory all instructions are 32-bit and must be word aligned. IA-32 processors, however, don’t have these restrictions for instruction alignment. All data types, like char, long, int, and float, all have the same sizes in ARM like in AI-32.

Altogether, ARM and IA-32 have a lot of similarities to them. Some of the main factors that make ARM different is that instead of specific registers that point to certain parts of the stack, it has 32-bit general purpose registers you have to access with load and store. It isn’t stack use heavy making it easier to move about when using your registers.

CITED SOURCES:

https://web.eecs.umich.edu/~prabal/teaching/eecs373-f10/readings/ARMv7-M_ARM.pdf

http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dai0274b/index.html