Scheme of 3rd int
Course: ARM embedded systems [18EC753] (open elective)
Sem: 7th sem
Dept. Of ETE, JNNCE, Shimoga

1a.

List out the problems faced by Compiler. Explain the Effect of local variable types by considering Checksum function.

Problems the compiler faces

- Test cases explicitly: No matter how advanced the compiler, it does not know whether N can be 0 on input or not. Therefore the compiler needs to test for this case explicitly before the first iteration of the loop.
- No alignment information: The compiler doesn't know whether the data array pointer is four-byte aligned or not. If it is four-byte aligned, then the compiler can clear four bytes at a time using an int store rather than a char store.
- Conservative compiler: The compiler must be conservative and assume all possible values for N and all possible alignments for data. Depends on the compiler vendor and compiler revision.

```
int checksum_v1(int *data)
{
  char i;
  int sum=0;

  for (i=0; i<64; i++)
  {
    sum += data[i];
  }
  return sum;
}
```

```
short checksum_v3(short *data)
{
  unsigned int i;
  short sum=0;

  for (i=0; i<64; i++)
  {
    sum = (short)(sum + data[i]);


  }
  return sum;
}
```

1b.Infer and explain the Efficient Use of c types:

■For local variables held in registers, don't use a char or short type unless 8-bit or 16-bit modular arithmetic is necessary. Use the signed or unsigned int types instead.
Unsigned types are faster when you use divisions.

■ For array entries and global variables held in main memory, use the type with the smallest size possible to hold the required data. This saves memory footprint. The ARMv4 architecture is efficient at loading and storing all data widths provided you traverse arrays by incrementing the array pointer. Avoid using offsets from the base of the array with short type arrays, as LDRH does not support this.

■ Use explicit casts when reading array entries or global variables into local variables, or writing local variables out to array entries. The casts make it clear that for fast operation you are taking a narrow width type stored in memory and expanding it to a wider type in the registers. Switch on implicit narrowing cast warnings in the compiler to detect implicit casts.

Scheme of 3rd int
Course: ARM embedded systems [18EC753] (open elective)
Sem: 7th sem
Dept. Of ETE, JNNCE, Shimoga

■ Avoid implicit or explicit narrowing casts in expressions because they usually cost extra cycles. Casts on loads or stores are usually free because the load or store instruction performs the cast for you.

■ Avoid char and short types for function arguments or return values. Instead use the int type even if the range of the parameter is smaller. This prevents the compiler performing unnecessary casts.

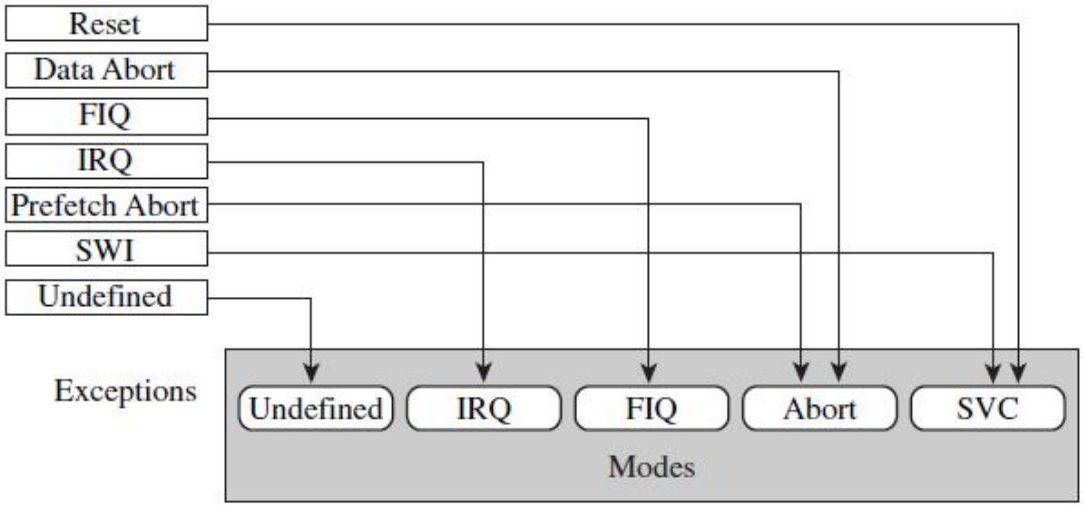
2a.List out all the ARM Processor Exceptions and associated Modes. Explain the core actions during mode change.

Exception Handling

- An exception is any condition that needs to halt the normal sequential execution of instructions.
  - Ex: when the ARM core is reset,
  - when an instruction fetch or memory access fails,
  - when an undefined instruction is encountered,
  - when a software interrupt instruction is executed, or
  - when an external interrupt has been raised.
- Exception handling is the method of processing these exceptions.
- Most exceptions have an associated software exception handler.
- software exception handler: a software routine that executes when an exception occurs. For instance, a Data Abort exception will have a Data Abort handler.
- The handler first determines the cause of the exception and then services the exception.
- Servicing takes place either within the handler or by branching to a specific service routine.
- The Reset exception is a special case since it is used to initialize an embedded system.

ARM processor exceptions and associated modes.

Exception	Mode	Main purpose
Fast Interrupt Request	*FIQ*	fast interrupt request handling
Interrupt Request	*IRQ*	interrupt request handling
SWI and Reset	*SVC*	protected mode for operating systems
Prefetch Abort and Data Abort	*abort*	virtual memory and/or memory protection handling
Undefined Instruction	*undefined*	software emulation of hardware coprocessors



Exceptions and associated modes.

2b.Infer and explain the Portability issues in ARM processors.

- Unaligned data pointers: Some processors support the loading of short and int typed values from unaligned addresses. A C program may manipulate pointers directly so that they become unaligned, for example, by casting a char * to an int *.
- Endian assumptions: C code may make assumptions about the endianness of a memory system, for example, by casting a char * to an int *.
- Function prototyping: The armcc compiler passes arguments narrow, that is, reduced to the range of the argument type. If functions are not prototyped correctly, then the function may return the wrong answer.
- Use of bit-fields: The layout of bits within a bit-field is implementation and endian dependent.
- Use of enumerations: Although enum is portable, different compilers allocate different numbers of bytes to an enum.
- Inline assembly: Using inline assembly in C code reduces portability between architectures.
- The volatile keyword: Use the volatile keyword on the type definitions of ARM memory-mapped peripheral locations. This keyword prevents the compiler from optimizing away the memory access. It also ensures that the compiler generates a data access of the correct type.

3a.Write short notes on vector table with typical vector table diagram.

vector table : a table of addresses that the ARM core branches to when an exception is raised.
These addresses commonly contain branch instructions of one of the following forms:

- B <address>—This branch instruction provides a branch relative from the pc.
- LDR pc, [pc, #offset]—This load register instruction loads the handler address from memory to the pc. The address is an absolute 32-bit value stored close to the vector table. Loading this absolute literal value results in a slight delay in branching to a specific handler due to the extra memory access. However, you can branch to any address in memory.
- LDR pc, [pc, #-0xff0]—This load register instruction loads a specific interrupt service routine address from address 0xffffff030 to the pc. This specific instruction is only used when a vector interrupt controller is present (VIC PL190).
- MOV pc, #immediate—This move instruction copies an immediate value into the pc. It lets you span the full address space but at limited alignment. The address must be an 8-bit immediate rotated right by an even number of bits.

Vector table and processor modes.

Exception	Mode	Vector table offset
Reset	SVC	+0x00
Undefined Instruction	UND	+0x04
Software Interrupt (SWI)	SVC	+0x08
Prefetch Abort	ABT	+0x0c
Data Abort	ABT	+0x10
Not assigned	—	+0x14
IRQ	IRQ	+0x18
FIQ	FIQ	+0x1c

3b.Explain the fundamental components of Embedded operating system.

- Initialization is the first code of the operating system to execute and involves setting up internal data structures, global variables, and the hardware. Initialization starts after the firmware hands over control. For hardware initialization an operating system sets up various control registers, initializes the device drivers, and, if the operating system is preemptive, sets up a periodic interrupt.
- Memory handling involves setting up the system and task stacks. The positioning of the stacks determines how much memory is available for either the tasks or the system. The decision as to here the system stack is placed is normally carried out during operating system initialization. Setting up the task stack depends upon whether the task is static or dynamic.
  - A static task is defined at build time and is included in the operating system image. For these tasks the stack can be set up during operating system initialization. For example, SLOS is a static-task-based operating system.
  - A dynamic task loads and executes after the operating system is installed and executing and is not part of the operating system image. The stack is set up when the task is created (for example, as in Linux).
- The scheduler is an algorithm that determines which task is to be executed next. There are many scheduling algorithms available. One of the simplest is called a round-robin algorithm—it activates tasks in a fixed cyclic order. Scheduling algorithms have to balance efficiency and size with complexity.
- The device driver framework—the mechanism an operating system uses to provide a consistent interface between different hardware peripherals. The frame work allows a standard and easy way of integrating new support for a particular peripheral into the operating system.

4a.Write short notes on firmware and bootloader.

The firmware is the deeply embedded, low-level software that provides an interface between the hardware and the application/operating system level software. It resides in the ROM and executes when power is applied to the embedded hardware system.
Firmware can remain active after system initialization and supports basic system operations.
The bootloader is a small application that installs the operating system or application onto a hardware target. The bootloader only exists up to the point that the operating system or application is executing, and it is commonly incorporated into the firmware.

Firmware execution flow.

Stage	Features
Set up target platform	Program the hardware system registers Platform identification Diagnostics Debug interface Command line interpreter
Abstract the hardware	Hardware Abstraction Layer Device driver
Load a bootable image	Basic filing system
Relinquish control	Alter the *pc* to point into the new image

4b.Explain the concept of link register offset with use of SUB and SUBS in handler code

- When an exception occurs, the link register is set to a specific address based on the current pc.
- For instance, when an IRQ exception is raised, the link register lr points to the last executed instruction plus 8.
- Care has to be taken to make sure the exception handler does not corrupt lr because lr is used to return from an exception handler.
- The IRQ exception is taken only after the current instruction is executed, so the return address has to point to the next instruction, or lr − 4.

Exception	Address	Use
Reset	—	*lr* is not defined on a Reset
Data Abort	*lr* − 8	points to the instruction that caused the Data Abort exception
FIQ	*lr* − 4	return address from the FIQ handler
IRQ	*lr* − 4	return address from the IRQ handler
Prefetch Abort	*lr* − 4	points to the instruction that caused the Prefetch Abort exception
SWI	*lr*	points to the next instruction after the SWI instruction
Undefined Instruction	*lr*	points to the next instruction after the undefined instruction

-

```
handler
        <handler code>
        ...
        SUBS    pc, r14, #4             ; pc=r14-4
```

```
handler
        SUB     r14, r14, #4            ; r14-=4
        ...
        <handler code>
        ...
        MOVS    pc, r14                 ; return
```

```
handler
        SUB     r14, r14, #4            ; r14-=4
        STMFD   r13!,{r0-r3, r14}       ; store context
        ...
        <handler code>
```