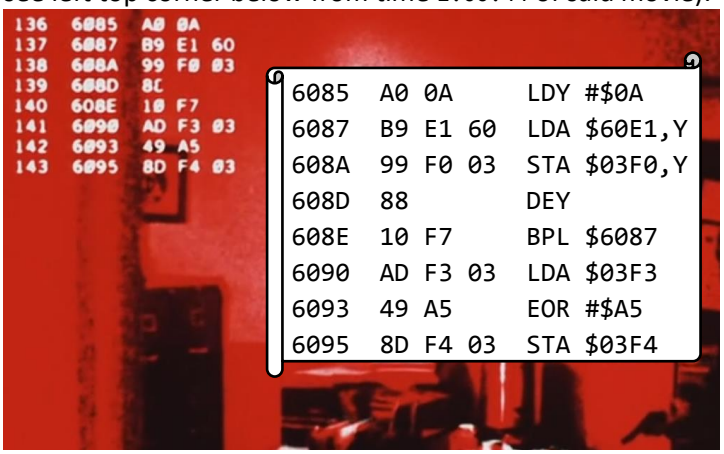
Write your answers to the special response sheet you received (with your name and photograph). If you are using more than a single sheet of paper for your answers, then mark each sheet with its number / total number of sheets you will hand over.

### Task 1

As we would like to be prepared in case *Skynet* seizes control of the world, we decided to analyze part of *T-800* cyborg machine code (that was by accident recorded in the 1984 in "*The Terminator*" movie by the Orion company – see left top corner below from time `1:00:44` of said movie):




We found out that T-800 is using a massive multicore 6502 CPU variant – it is an 8-bit CPU with 16-bit physical address space and accumulator architecture: it contains register `A`, two additional general auxiliary registers `X` and `Y`, typical flags register, and one 16-bit register = the `PC` register. CPU's ISA includes the following instructions: `LDA` (load `A`), `LDY` (load `Y`), `STA` (store `A`), `EOR` (exclusive or), `BPL` (branch if plus) – all having a single explicit argument, and `DEY` (decrement `Y`) instruction with no arguments. All instructions have usual semantics. We disassembled the T-800 machine code into standard 6502 assembler – see the right part of image above – that is using the following conventions: `#$xx` is an 8-bit immediate argument, `$xxxx` = direct absolute address, expression `$xxxx,Y` = direct absolute address computed as `($xxxx + Y)`, `$xxxx` being an immediate and `Y` is the auxiliary CPU register. Rewrite the T-800 code into Pascal without using inline assembler in a way a human programmer would code it. Devise a suitable name for all global variables used in the original code and write their complete declaration (**think about data organization in memory**).

---

### Task 2

Design a complete HCI of a hard drive controller, i.e. define all necessary registers, all commands it needs to support, and a communication protocol towards drivers. The controller has to support any hard drive with `512 B` sectors and up to `H` heads, `C` tracks, `S` sectors per track, i.e. up to `TS` total sectors: `TS = H * C * S`. The controller has to support reading and writing data in units of sectors and at least PIO mode.

---

### Task 3

Assume a computer with Intel 430LX chipset that includes the Intel 82434LX integrated DRAM memory controller and Host/PCI bridge. A 32-bit CPU Intel Pentium P54CS with internal clock frequency of 133 MHz and 16 KiB cache is connected to a 64-bit FSB with 32-bit address space and 66 MHz clock frequency. The zeroth 32-bit 33 MHz PCI bus directly connected to the Host/PCI bridge hosts also a sound card supporting both PIO and DMA bus master transfers.

We need to transfer 1 MiB of data from the sound card into this computer's main memory. Do an educated guess, which of the PIO or DMA transfers should be faster. Calculate approximately how many times is the faster variant faster than the slower one taking into account the most ideal (but realistic) situation that can happen. Choose appropriately any additional constants necessary for the calculation.

---

### Task 4

Design a 16-bit parallel system bus supporting 32-bit address space and regular read and write transfers (no need to support burst transfers). Describe and explain all signals necessary for such a bus to function correctly. Also draw timing diagrams for single value read and single value write – you can combine multiple signals into a single row in the diagram when reasonable.

---

### Task 5

Assume the following function written in Pascal that should compare two UTF-16 zero-terminated strings for equality, i.e. returning `true`, if they both represent the same piece of text:

```
type
  PUtf16 = ^word;
function Equivalent(
  s1 : PUtf16; s2 : PUtf16) : boolean;
begin
  Equivalent := true;
  while true do begin
    if s1^ <> s2^ then begin
      Equivalent := false;
      break;
    end;
    if s1^ = 0 then break;
    s1 := s1 + 1; s2 := s2 + 1;
  end;
end;
```

Does this function work correctly for all possible input string combinations with respect to standard Unicode semantics? Explain in detail why.

---

### Task 6

Explain principle of *differential transfer* and state all pros and cons. Draw and describe an example timing diagram for some chosen value.

---

### Task 7

Write in Pascal function Conv with the below stated declaration, that will translate a **non-zero *number*** in floating-point type *single* (type *single* is a 32-bit floating-point number defined by IEEE 754 standard, i.e. it has a normalized mantissa [significand] with hidden 1 occupying lowest 23 bits of the value, followed by 8 bit exponent in bias +127 format, and the last bit [MSb] is a sign bit) into 16-bit **signed** integer type `integer`. The function should return integral part of the original real number. If the original value is too small (in between 1 and -1) or the integral part is out of range of the `integer` type, the function should return value `0`. Use only Pascal's integer arithmetic in all of your code, and consider taking advantage of bitwise operations supported by Pascal.

```
function Conv(flt32 : longword) : integer;
```

---

### Task 8

Assume we are programming support for basic synchronization primitives as API functions of our OS kernel supporting preemptive multithreading and targeting x86/IA-32 processor architecture. Our kernel supports single processor systems only.

Write in Pascal a `Lock` record declaration that will represent state of a standard lock with usual semantics. Also provide an implementation of `Enter` (will try to lock a provided lock for the calling thread – it should use only passive waits if waiting is necessary) and `Exit` (will unlock a provided lock for the calling thread) functions/procedures using the declared `Lock` record. **Document all code in your implementation and especially explain necessity of the core parts of your code.** In your code you can use any reasonable API functions that would be provided by a typical OS scheduler to the rest of such OS kernel or applications – do not implement such functions, but provide a short explanation of their behavior.

---

### Task 9

If a regular program written in Pascal is compiled, does the entrypoint of the resulting executable file point to the 1st instruction generated from the program's main program, i.e. from **`begin` … `end.`**)? Explain in detail. If not, explain what code will run before the main program and what are its tasks.

---

### Task 10

Write in Pascal a program that would take as its first argument [available as a result of standard Pascal function `ParamStr(1)`] a name of a file containing binary image of a disk partition formatted with the FAT16 file system. The goal of your program is to find description of the root directory in the disk image and print out list of all files contained in it – for every file in the root directory your program should print file's name and extension separated by a comma, information for each file should be on a separate line. The FAT16 file system supports filenames of up to 8 characters and up to 3 characters of file extension. If the file name is shorter than 8 chars or extension is shorter than 3, they should be padded by spaces from right – e.g. if the root directory contains the following four files `A.TXT`, `COMPUTER.EXE`, `1STLABEL`, and `HELLO.A`, your program should print:

```
A       .TXT
COMPUTER.EXE
1STLABEL.
HELLO   .A
```

Partition's boot sector contains the basic information about FAT16 file system's structure, and has the following format:

| Offset | \|B\| | Description |
|---|---|---|
| `0x000` | 3 | First instruction executed as part of the boot process when booting from this partition (typically `$EB $?? $90`) |
| `0x003` | 8 | ASCII string: name of program that formatted this partition |
| `0x00B` | 2 | Number of bytes per sector (typically `512`) |
| `0x00D` | 1 | Number of sectors per cluster |
| `0x00E` | 2 | Number of reserved sectors = number of sectors (including this boot sector) preceding the first copy of the FAT table |
| `0x010` | 1 | Number of identical FAT copies (typically `2`) |
| `0x011` | 2 | Maximal number of records in the root directory |
| `0x013` | 2 | Reserved (filled with `0`) |
| `0x015` | 1 | Reserved |
| `0x016` | 2 | No. of sectors used by a single FAT copy |

The boot sector is followed by a variable amount of reserved sectors, which are then followed by copies of FAT table. The FAT tables are directly followed by the root directory data – **beware:** the root directory is the only file in the FAT16 file system that has a fixed beginning (first data sector of the root directory = first sector directly after the last sector occupied by the last FAT copy, i.e. "no. of reserved sectors" + "no. of FAT copies" * "no. of sectors per single FAT"). All of the root directory's data sectors are preallocated continuously one after other when the partition was formatted and are **not** managed by the FAT table. Total number of root directory data sectors is implied by the maximal number of its records, as defined in the boot sector. Every directory record looks as:

| Offset | \|B\| | Description |
|---|---|---|
| `0x00` | 8 | ASCII string: file name (padded from right by spaces = ASCII code `$20`). Value `0x00` or `0xE5` in the byte `0` implies an unused directory record → it does not have any meaningful info stored in its bytes 1-31. |
| `0x08` | 3 | ASCII string: file extension (padded from right by spaces). Files without an extension have this field filled with 3 spaces. |
| `0x0B` | 1 | Flags: bit 4: `0` = a file, `1` = a directory |
| `0x0C` | 10 | Reserved |
| `0x16` | 2 | Last file modification time |
| `0x18` | 2 | Last file modification date |
| `0x1A` | 2 | First data cluster number of this file |
| `0x1C` | 4 | File length in bytes |

Note: your program should support disk images with up to 4 kB sectors.