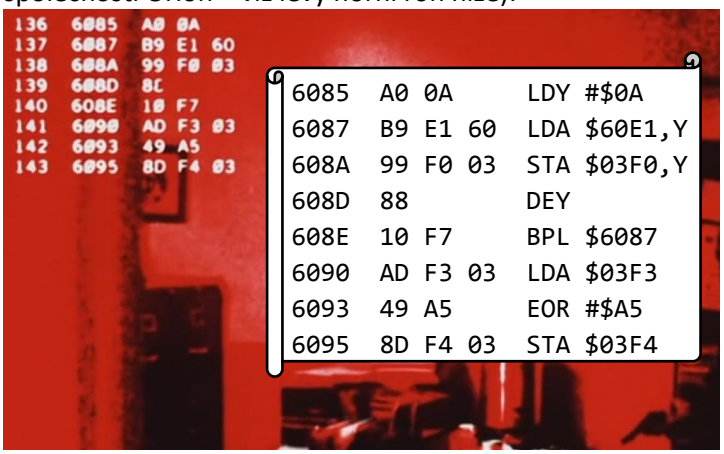
Odpovědi pište na zvláštní odpovědní list s vaším jménem a fotografií. Pokud budete odevzdávat více než jeden list s řešením, tak se na 2. a další listy nezapomeňte podepsat. Do zápatí všech listů vždy napište i/N (kde i je číslo listu, N je celkový počet odevzdaných listů).

Otázka č. 1

Chtěli bychom se připravit na možnost ovládnutí světa *Skynet*em, proto jsme se rozhodli analyzovat část strojového kódu kyborga *T-800* (který byl mimoděk v roce 1984 zachycen v čase `1:00:44` ve filmu „*The Terminator*“ společnosti Orion – viz levý horní roh níže):



Zjistili jsme, že se v T-800 používá masivně multijádrová varianta procesoru 6502 – je to 8-bitový procesor s akumulátorovou architekturou a s 16-bit fyzickým adresovým prostorem: má registr A, dva další obecné pomocné registry X, a Y, běžný příznakový registr, a registr PC, který je jako jediný 16-bitový. CPU má mimo jiné instrukce `LDA` (load A), `LDY` (load Y), `STA` (store A), `EOR` (exclusive or), `BPL` (branch if plus) – všechny s jedním explicitním argumentem, a instrukci `DEY` (decrement Y) bez argumentů. Instrukce mají běžnou sémantiku. Strojový kód z T-800 jsme disassemblovali do standardního 6502 assembleru – viz pravý výřez obrázku výše – který používá následující konvence: `#$xx` znamená 8-bitový immediate argument, `$xxxx` = přímá absolutní adresa, výraz `$xxxx,Y` = přímá absolutní adresa spočítaná jako `($xxxx + Y)`, kde `$xxxx` je immediate a `Y` je pomocný registr procesoru. Zapište bez použití inline assembleru v Pascalu uvedený kód T-800 tak, jak by ho napsal lidský programátor. Pro všechny v původním kódu použité globální proměnné si vymyslete nějaký název a napište jejich kompletní deklaraci (**zamyslete se nad organizací dat v paměti**).

Otázka č. 2

Navrhněte kompletní HCI řadiče pevných disků – popište všechny navržené registry, způsob komunikace s řadičem z ovládajícího SW, a všechny příkazy, které bude muset podporovat. Řadič má podporovat připojení libovolného disku s `512` B sektory a s maximálně `H` hlavičkami, `C` stopami, a `S` sektory na stopu, tj. maximálně s celkovým počtem `TS` sektorů: `TS = H * C * S`. Řadič má umožňovat čtení a zápis dat na pevný disk po celých sektorech, a přenos dat minimálně pomocí PIO režimu.

Otázka č. 3

Předpokládejte, že máme počítač s chipsetem Intel 430LX jehož součástí je integrovaný řadič pamětí DRAM a Host/PCI bridge Intel 82434LX. Na 64-bitové FSB s 32-bitovým adresovým prostorem (taktovací frekvenci 66 MHz) je připojen 32-bitový procesor Intel Pentium P54CS s vnitřní taktovací frekvencí 133 MHz a s 16 KiB pamětí cache. Na nulté 32-bitové 33 MHz PCI sběrnici připojené přímo k Host/PCI bridge je také připojena zvuková karta s podporou PIO i DMA bus master přenosů.
Na tomto počítači budeme potřebovat ze zvukové karty do operační paměti přenést 1 MiB dat. Proveďte v této situaci kvalifikovaný odhad, zda bude rychlejší data přenést pomocí PIO nebo DMA režimu. Přibližně spočítejte, o kolik bude která varianta rychlejší, pokud budete brát v potaz pro váš výpočet nějakou nejideálnější situaci, která by na uvedeném systému pro vás mohla nastat. Případné další konstanty, které byste potřebovali znát, si vhodně zvolte.

Otázka č. 4

Navrhněte 16-bitovou paralelní systémovou sběrnici s podporou pro 32-bitový adresový prostor a možností běžného čtení a zápisů dat (bez podpory burst přenosů). Rozhodněte a vysvětlete, minimálně jaké všechny vodiče vaše sběrnice bude muset mít, a nakreslete časový diagram čtení a časový diagram zápisu nějaké hodnoty. Vodiče, u kterých to dává smysl, můžete v obrázku seskupit do jednoho řádku diagramu.

Otázka č. 5

Předpokládejte následující funkci v Pascalu, která má na rovnost srovnat dva UTF-16 zero-terminated řetězce a vrátit `true`, pokud jsou oba řetězce ekvivalentní (tj. reprezentují ten samý 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;
```

Funguje tato fce správně pro všechny možné textové řetězce vzhledem k běžné sémantice Unicode? Detailně vysvětlete proč.

Otázka č. 6

Vysvětlete princip *diferenciálního přenosu*, a uveďte jeho výhody, případně nevýhody. Nakreslete také časový diagram přenosu nějaké hodnoty a popište na něm jaké.

Otázka č. 7

Naprogramujte v Pascalu funkci Conv s níže uvedeným prototypem, která převede **nenulové *číslo*** ve floating-point formátu *single* (typ *single* je 32-bitové floating-point číslo dle standardu IEEE 754, tj. mantisa je normalizována se skrytou 1 a zabírá spodních 23 bitů, pak následuje 8-bitový exponent uložený ve formátu s posunem [bias] +127, a poslední bit, tedy MSb, je znaménkový bit) do 16-bitového **znaménkového** celočíselného typu `integer`. Funkce má vrátit celou část původního reálného čísla. Pokud je původní hodnota příliš malá (mezi 1 a -1), nebo je celá část původní hodnoty mimo rozsah typu `integer`, má funkce vrátit hodnotu 0. Celý výpočet/zpracování zapište jen s využitím celočíselné aritmetiky Pascalu, a zvažte možnost použít pro výpočet bitové operace podporované v Pascalu.

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

Otázka č. 8

Předpokládejte, že programujeme podporu pro základní synchronizační primitiva jako API funkce jádra operačního systému s preemptivním přepínáním vláken určeného pro architekturu procesorů x86/IA-32. Naše jádro bude zatím podporovat pouze jednoprocesorové systémy.
Naprogramujte v Pascalu deklaraci záznamu `Lock`, který bude v jádře našeho OS reprezentovat stav jednoho standardního zámku s běžnou sémantikou. Dále s jeho použitím naprogramujte API funkci/proceduru `Enter` (která se pro volající vlákno pokusí zámek zamknout – pro čekání na odemčení zámku využijte pouze pasivní čekání), a funkci/proceduru `Exit`, která za volající vlákno zámek odemkne. **Vaši implementaci okomentujte, a vysvětlete důležitost hlavních celků vašeho kódu.**
V implementaci můžete využít API funkce, které by v běžném OS poskytoval plánovač pro zbytek jádra, případně pro aplikace – takové funkce nemusíte programovat, ale napište k nim stručné vysvětlení jejich chování.

Otázka č. 9

Přeložíte-li běžný program napsaný v Pascalu, bude entrypoint spustitelného souboru ukazovat na 1. instrukci vzniklou z vašeho hlavního programu (tj. z **`begin` … `end.`**)? Detailně vysvětlete. Pokud ne, tak vysvětlete, jaký kód poběží „před hlavním programem“ a co bude dělat.

Otázka č. 10

Naprogramujte v Pascalu program, který jako 1. argument na příkazové řádce – dostupný jako výsledek běžné Pascal funkce `ParamStr(1)` – dostane jméno souboru s binárním obrazem diskového oddílu naformátovaného souborovým systémem FAT16. Úkolem programu je najít v obrazu popis kořenového adresáře a vypsat seznam všech souborů, které obsahuje – pro každý soubor obsažený v kořenovém adresáři má vypsat jeho jméno a příponu oddělené tečkou, informace o každém souboru má být na zvláštním řádku. Souborový systém FAT16 podporuje maximálně 8 znaková jména a maximálně 3 znakové přípony souborů – pokud je jméno souboru kratší než 8 znaků (resp. přípona kratší než 3), tak se mají ve výpisu zprava dorovnat mezerami, např. pokud kořenový adresář obsahuje soubory `A.TXT`, `COMPUTER.EXE`, `1STLABEL`, a `HELLO.A`, má program vypsat:

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

Boot sektor oddílu obsahuje základní informace o struktuře souborového systému FAT16, a má následující formát:

Offset	|B|	Popis
0x000	3	První instrukce provedená při bootování z tohoto oddílu (typicky $EB $?? $90)
0x003	8	ASCII řetězec se jménem programu, který tento oddíl zformátoval
0x00B	2	Počet bytů na sektor (typicky 512)
0x00D	1	Počet sektorů na cluster
0x00E	2	Počet rezervovaných sektorů = počet sektorů (včetně tohoto boot sektoru), které se nacházejí před první kopií FAT tabulky
0x010	1	Počet identických kopií FAT tabulky (typ. 2)
0x011	2	Maximální počet záznamů v kořenovém adresáři
0x013	2	Rezervováno (vyplněno 0)
0x015	1	Rezervováno
0x016	2	Počet sektorů, který zabírá jedna kopie FAT tabulky

Ve vašem programu můžete počítat s tím, že největší velikost sektoru (počet bytů na sektor), na kterou můžete narazit ve zpracovávaném diskovém obrazu, jsou 4 kB.
Po boot sektoru následuje proměnné množství rezervovaných sektorů, pak následují kopie FAT tabulky, pak hned následují data kořenového adresáře – **pozor:** kořenový adresář je u FAT16 jediný soubor, který má pevný začátek (první datový sektor kořenového adresáře = první sektor přímo následující hned za posledním sektorem poslední kopie FAT tabulky, tj. „počet rezervovaných sektorů“ + „počet FAT“ * „počet sektorů jedné kopie FAT“), a veškeré jeho datové sektory jsou při formátování oddílu předalokovány kontinuálně za sebou a **nejsou** spravovány FAT tabulkou. Počet sektorů kořenového adresáře je dán maximálním počtem jeho záznamů – viz položka v boot sektoru. Jeden záznam adresářové položky má formát:

Offset	|B|	Popis
0x00	8	ASCII řetězec se jménem souboru (zprava dorovnaný mezerami = ASCII kód $20). Pokud je 0. byte roven 0x00 nebo 0xE5, tak je tento adresářový záznam nepoužitý a byty 1-31 nenesou platné informace.
0x08	3	ASCII řetězec přípony souboru (zprava dorovnaný mezerami). Soubory bez přípony mají tuto položku vyplněnou 3 mezerami.
0x0B	1	Sada příznaků: bit 4: 0 = soubor, 1 = adresář
0x0C	10	Rezervováno
0x16	2	Čas poslední modifikace souboru
0x18	2	Datum poslední modifikace souboru
0x1A	2	Číslo prvního datového clusteru souboru
0x1C	4	Délka souboru v bytech