Organizacja i architektura komputerów

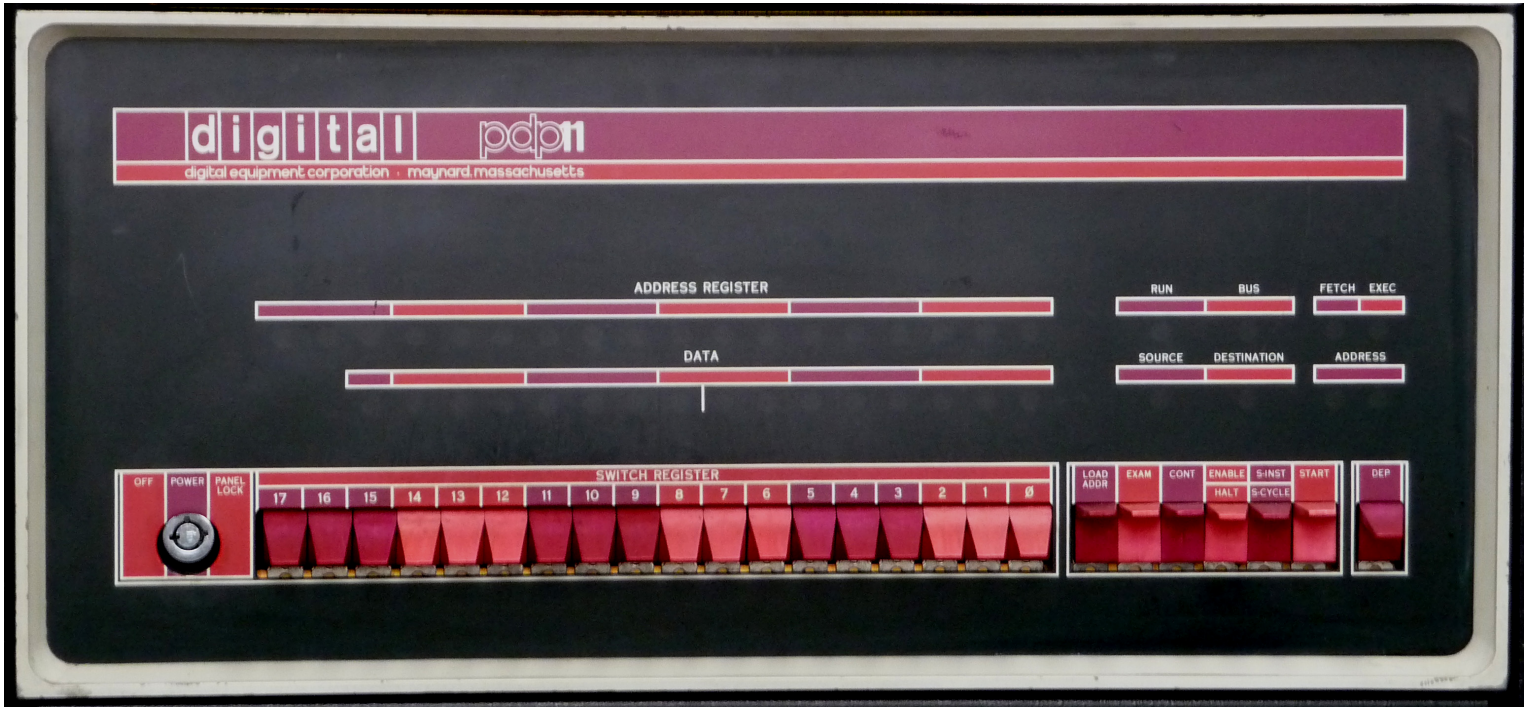
Asembler & GDB cd

Piotr Patronik

29 lutego 2016

Procesor jako maszyna

PDP-11 (1971)



Procesor jako maszyna (2)

Altair 8800 (1975)

Procesor jako maszyna (3)

- Operacje panelu (monitora)
  - Załaduj liczbę do pamięci (ustaw przełączniki+zatrzaśnij)
  - Wykonaj rozkaz (liczbę) z pamięci (pojedynczo)
  - Wykonuj rozkazy ciągiem (wykonaj program)
- Możliwe ładowanie programu z pamięci zewnętrznej
  - Taśmy papierowej...
  - Kart dziurkowanych
  - Taśmy magnetycznej
  - Dysku twardego
- Stan maszyny = rejestry+pamięć
  - Pełna obsługa z panelu
- Prawdziwy programista nie potrzebuje klawiatury...

Program – kod

test.s

```
# vim: syntax=gas

.text

hello: .ascii "Something.\n"
hello_len = . - hello

.global _start
_start:

mov $4, %eax #(sys)write
mov $1, %ebx #stdout
mov $hello, %ecx
mov $hello_len, %edx
int $0x80


mov $1, %eax #(sys)exit
mov $0, %ebx #exit code 0
int $0x80
```

Program – sprawdzenie działania

Asemblacja, linkowanie, uruchomienie

```
pepe@lenovo:/tmp$ as -o test.o test.s
pepe@lenovo:/tmp$ ld -o test test.o
pepe@lenovo:/tmp$ ./test
Something.
pepe@lenovo:/tmp$
```

Program = liczby w pamięci

test2.s

```
# vim: syntax=gas

.text

.global _start
_start:

.byte 0xb8, 0x04, 0x00, 0x00, 0x00, 0xbb
.byte 0x01, 0x00, 0x00, 0x00, 0xb9, 0x9a
.byte 0x00, 0x40, 0x00, 0xba, 0x0b, 0x00
.byte 0x00, 0x00, 0xcd, 0x80, 0xb8, 0x01
.byte 0x00, 0x00, 0x00, 0xbb, 0x00, 0x00
.byte 0x00, 0x00, 0xcd, 0x80, 0x53, 0x6f
.byte 0x6d, 0x65, 0x74, 0x68, 0x69, 0x6e
.byte 0x67, 0x2e, 0x0a
```

- Asemblacja automatyczna (program)
- Możliwa też asemblacja ręczna
  - Lista rozkazów (i kodowanie rozkazów!)

. . . – sprawdzenie działania

Asemblacja, linkowanie, uruchomienie

```
pepe@lenovo:/tmp$ as -o test2.o test2.s
pepe@lenovo:/tmp$ ld -o test2 test2.o
pepe@lenovo:/tmp$ ./test2
Something.
pepe@lenovo:/tmp$
```

- (można też z ręki napisać plik ELF. . .)

GDB jako monitor

Panel przedni

- Załadowanie programu do pamięci (`gdb ./test`)
- Sprawdzenie stanu maszyny (`info registers`, `x`, `print`)
- Wykonanie pojedynczego rozkazu (`stepi`)
- Wykonanie programu (`run`/`continue`)

Model maszyny – rejestry

- Rejestr przechowuje liczby
  - Jaki jest rozmiar rejestru 8051/i386/x64?
- Rejestr NIE przechowuje: obiektów, stringów, wskaźników
- Rejestr to przerzutnik. . . jaki?
- Można do niego załadować lub wykorzystać jego wartość
  - Dodać, odjąć, pomnożyć. . .
  - Wykonać operację logiczną: AND, OR, NOT. . .
  - Przesłać do innego rejestru

Sprawdzenie (za)wartości rejestru

- Ładujemy program do pamięci (z taśmy...)
  - Możemy też wklepać po jednej liczbie...
- Ustawiamy licznik rozkazów na pierwszy adres

```
pepe@lenovo:/tmp$ gdb -q test
Reading symbols from test...(no debugging symbols found)...done.
(gdb) break _start
Breakpoint 1 at 0x400078
(gdb) run
Starting program: /tmp/test

Breakpoint 1, 0x0000000000400078 in _start ()
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x0000000000400078 <+0>:     mov    $0x4,%eax
   0x000000000040007d <+5>:     mov    $0x1,%ebx
   0x0000000000400082 <+10>:    mov    $0x40009a,%ecx
   0x0000000000400087 <+15>:    mov    $0xb,%edx
   0x000000000040008c <+20>:    int    $0x80
```

- Czego się spodziewamy po wykonaniu rozkazu `mov $0x4, %eax`?

Rejestry cd

- Sprawdzamy...

```
(gdb) stepi
0x000000000040007d in _start ()
(gdb) info registers
rax            0x4      4
rbx            0x0      0
rcx            0x0      0
rdx            0x0      0
(gdb) print $eax
$1 = 4
(gdb) stepi
0x0000000000400082 in _start ()
(gdb) print $ebx
$2 = 1
```

(Częściowe) wnioski

- Procesor jest (prostą) maszyną
  - Wykonuje program złożony z rozkazów
  - Rozkazy są liczbami
  - Rozkazy operują na rejestrach
- GDB jest monitorem procesora
  - Umożliwia krokowe wykonanie programów
  - Umożliwia analizowanie zawartości rejestrów