Systemy komputerowe (W05)

Arytmetyka CPU i programowanie CPU

A.Pruszkowski

Arytmetyka CPU

Systemy Komputerowe

- Wielkości typów danych
  - char (int8_t)
    - 8 bitów
      - 1 bit znaku
      - 7 bitów wartości
    - min: -128
    - max: 127
  - unsigned char (uint8_t)
    - 8 bitów
      - 8 bitów wartości
    - min: 0
    - max: 255

Systemy Komputerowe

- **Wielkości typów danych**
  - **short (int16_t)**
    - **16 bitów**
      - **1 bit znaku**
      - **15 bitów wartości**
    - **min: -32 768**
    - **max: 32 767**
  - **unsigned short (uint16_t)**
    - **16 bitów**
      - **16 bitów wartości**
    - **min: 0**
    - **max: 65 535**

Systemy Komputerowe

- **Wielkości typów danych**
  - **long (int32_t)**
    - **32 bitów**
      - **1 bit znaku**
      - **31 bitów wartości**
    - **min: -2 147 483 648**
    - **max: 2 147 483 647**
  - **unsigned long (uint32_t)**
    - **32 bitów**
      - **32 bitów wartości**
    - **min: 0**
    - **max: 4 294 967 296**

Systemy Komputerowe

- **Wielkości typów danych**
  - **long long (int64_t)**
    - **64 bitów**
      - **1 bit znaku**
      - **63 bitów wartości**
    - **min:** **-**9 223 372 036 854 775 808
    - **max:** 9 223 372 036 854 775 807
  - **unsigned long long (uint64_t)**
    - **64 bitów**
      - **64 bitów wartości**
    - **min:** 0
    - **max:** 18 446 744 073 709 551 616

Systemy Komputerowe

- **Wielkości typów danych (IEEE 754)**
  - **float (32 bitów)**
    - **1 bit znaku (S)**
    - **8 bitów wykładnika (E)**
    - **23 bitów mantysy (M)**
  - **min:** ~ +/- 10^{-38}
  - **max:** ~ +/- 10^{38}
  - **double (64 bitów)**
    - **1 bit znaku (S)**
    - **11 bitów wykładnika (E)**
    - **52 bitów mantysy (M)**
  - **min:** ~ +/- 10^{-308}
  - **max:** ~ +/- 10^{308}

Jak wyliczać wartość

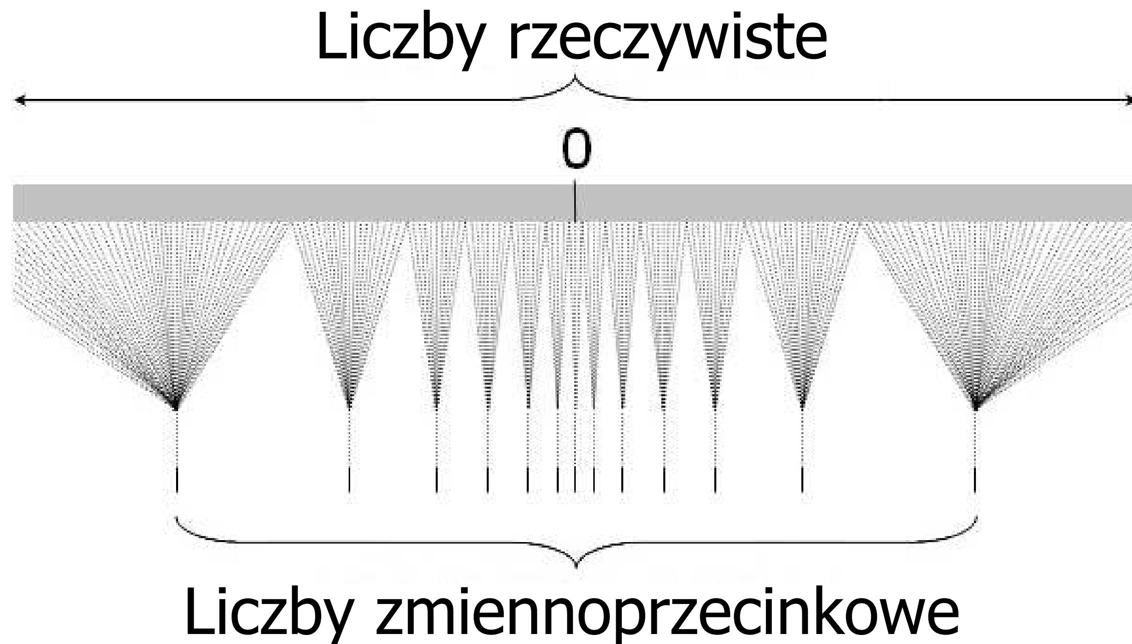
$$x = S * M * 2^{E}$$

Wartości szczególne (np.: float):

- Nieskończoność („quiet NaN”)
  - S=0, E=0xFF => - ∞
  - S=1, E=0xFF => + ∞
- lub
  - E=0xFE => jako nie reprezentowalna wartość („signaling NaN”)

Systemy Komputerowe

- **Problem obliczeń z użyciem liczb zmiennoprzecinkowych**
  - **Liczba zmiennoprzecinkowa reprezentuje wiele liczb rzeczywistych**
    - Problem brak przechodniości: $F_1=f(R_1)$ ale $R_2=f(F_1)$ gdzie $|R_1-R_2|=d$, $d=f(F_1)$
      - konsekwencja utrata dokładności (np.: przy dodawaniu małych wartości do dużych)
      - rozwiązanie: dodaj wpierw małe wartości, a potem ich sumę do dużych wartości



Zależność między liczbami rzeczywistymi a zmiennoprzecinkowymi

Źródło: http://jasss.soc.surrey.ac.uk

Systemy Komputerowe

- **Wielkości typów danych - problemy**
  - **typ: int**
    - **Implementacja zależna od kompilatora**
      - **jego architektury domyślnego traktowania tego typu**
      - **użytych opcji wywołania procesu kompilacji**
      - **kompilator może zatem potraktować „int" jako zmienną o liczbie bitów: 8,16,32,64**
  - **Zaleca się stosowanie typów o ustalonej liczbie bitów, za pomocą definicji nowych typów, np.:**

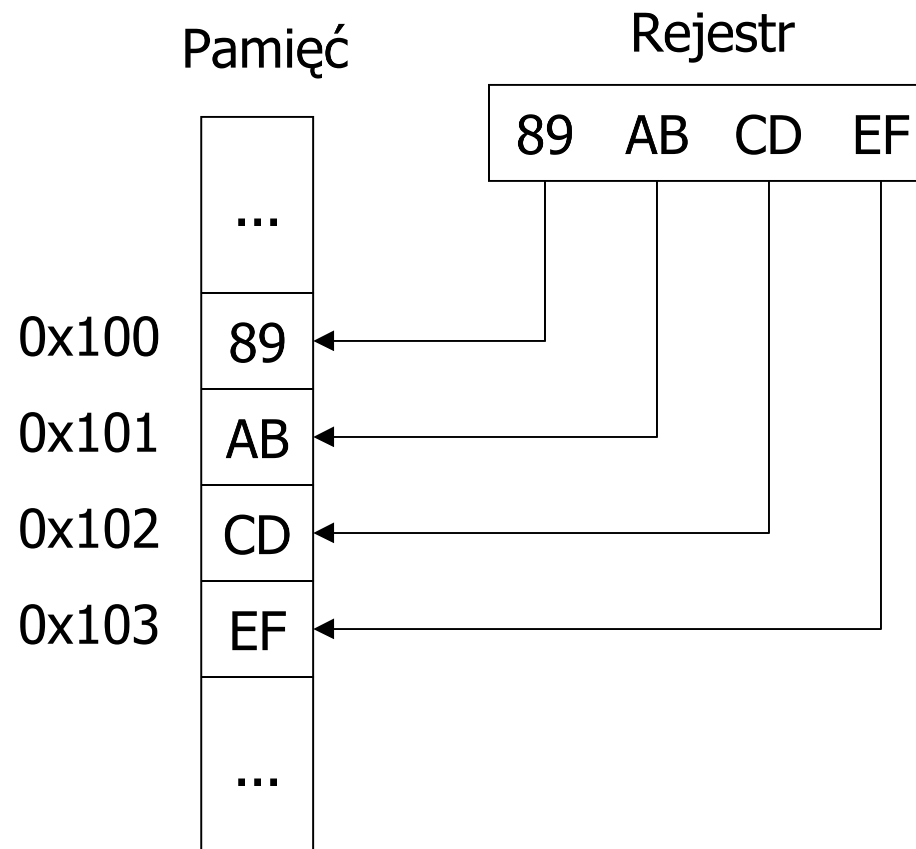
```
typedef unsigned char uint8_t;

typedef unsigned short uint16_t;

typedef unsigned long uint32_t;
```

Systemy Komputerowe

- **Kolejność bitów**
  - **Bajt (1B)**
    - **Tu wszystkie systemy potrafią się porozumiewać bezbłędnie!**
  - **Słowo (2B), podwójne słowo (4B), ...**
    - **Tu systemy muszą „rozmawiać” w jednej wybranej notacji:**
      - **grubo-końcówkowej (big endian)**
      - **cienko-końcówkowej (little endian)**
  - **Gdzie występuje problem?**
    - **Operacje zwykłe:** ***lokalno - lokalne***
      - **Nie**
    - **Operacje dyskowe:** ***lokalno - "potencjalnie zdalne"*** **(np.: pendrive)**
      - **Tak**
    - **Operacje sieciowe:** ***zdalne***
      - **Tak**

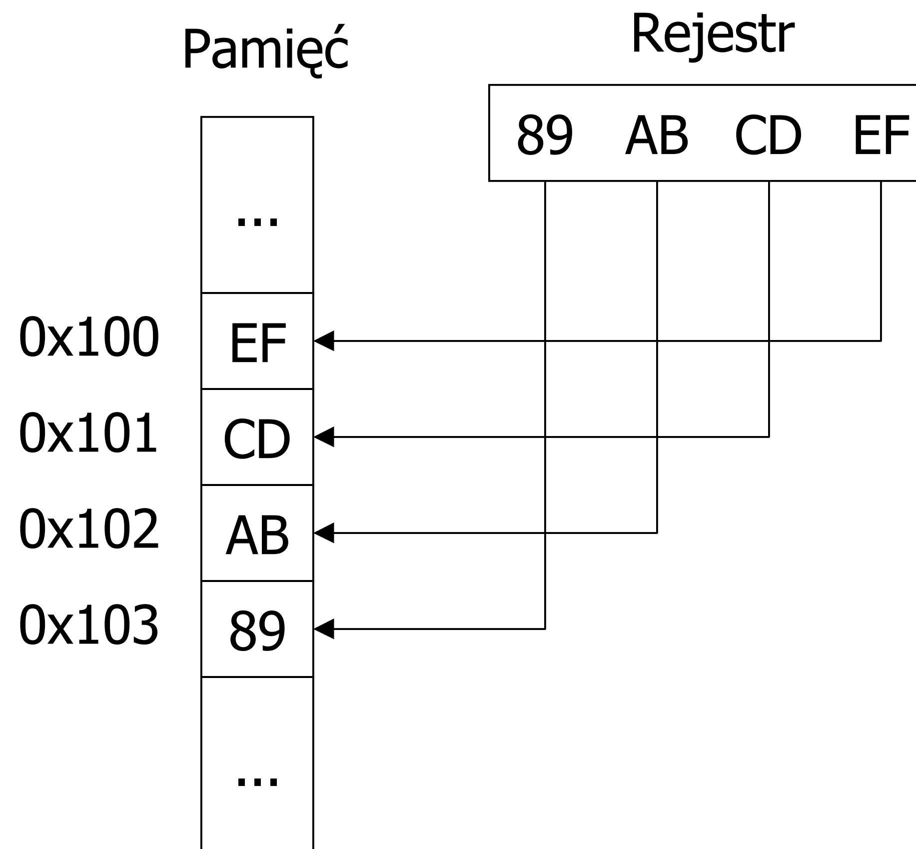
Systemy Komputerowe

- **Kolejność bitów, cd .**
  - **Grubo-końcówkowa (big endian)**



Systemy Komputerowe

- **Kolejność bitów, cd .**
  - **Cienko-końcówkowa (little endian)**



Systemy Komputerowe

- **Różnica między wartością a znakiem/znakami**
  - Wartość 65 to ‘A’, jak zatem pokazać zawartość rejestru R13 równą 65?
  - Lub wartość 1234 to w pamięci zapis 0x04D2
  - Aby pokazać te wartości (VAL) niezbędna jest konwersja wartości na postać tekstową zgodną z notacją DEC / HEX / OCT / BIN
    - konwersje VAL <-> HEX, VAL <-> OCT, VAL <-> BIN - trywialne
    - konwersje VAL <-> DEC - nieco bardziej skomplikowane
- **Zalety i wady posługiwania się różnymi systemami liczbowymi**
    - BIN: szybko widać które bity mają jaką wartość, postać rozwlekła
    - HEX: postać zwięzła, przy pewnej wprawie widać które bity mają jaką wartość, szybka konwersja do OCT/BIN
    - OCT: rzadziej stosowana (np.: opis praw dostępu do plików w Linux)
    - DEC: postać naturalna człowiekowi, konwersje na inne postaci mocno utrudnione

Systemy Komputerowe

- **Łańcuchy danych**
  - Model „PASCAL”
    - <liczba znaków><znak$_1$><znak$_2$>...<znak$_{\text{liczba znaków-1}}$>
      - Zalety: Przechowywane mogą być dowolne znaki
      - Wady: Łańcuch ma skończoną długość (wersje o długości 255 lub 65535 znaków), Brak zgodności między wersjami (!)
  - Model „C”
    - <znak$_1$><znak$_2$>...<znak$_N$><znak$_{\text{specjalny}}$>
      - Zalety: Długość łańcucha niemal nieograniczona (ogranicza wielkość pamięci)
      - Wady: Wśród znaków nie można przechowywać znaku specjalnego
  - Inne modele?

Systemy Komputerowe

- **Wartość liczby - NKB**
  - **Dla N+1 bitowej liczby A**
  - **Gdzie:**
    - $A = (a_N, a_{N-1}, \ldots, a_1, a_0)_{NKB}$
    - a_x = wartość bitu x
  - **Wartość takiej liczby to**

$$A = (-1)^{a_N} * \sum_{i=0}^{N-1} a_i * 2^i$$

  - **Np.: N+1 = 4**
    - 0 101 -> wartość: $-1^0 * (2^2 + 2^0) = 1 * (4 + 1) = 5$
    - 1 101 -> wartość: $-1^1 * (2^2 + 2^0) = -1 * (4 + 1) = -5$
  - ***Wada: podwójne kodowanie wartości 0***

Systemy Komputerowe

- **Wartość liczby - U2**
  - **Dla N+1 bitowej liczby A**
  - **Gdzie:**
    - $A = (a_N, a_{N-1}, \ldots, a_1, a_0)_{NKB}$
    - a_x = wartość bitu x
  - **Wartość takiej liczby to**

$$A = (-a_N) * 2^N + \sum_{i=0}^{N-1} a_i * 2^i$$

  - **Np.: N+1 = 4**
    - 0 101 -> wartość: $-0*2^3 + 2^2 + 2^0 = 4 + 1 = 5$
    - 1 101 -> wartość: $-1*2^3 + 2^2 + 2^0 = -8 + 4 + 1 = -3$
  - **Tu „0" staje się wartością dodatnią ale jest kodowane tylko raz!**

- **Flaga *O* (nadmiar)**
  - **Oznacza ze wynik operacji nie „zmieścił się" w reprezentowalnej postaci**
  - ***O* nazywane tez**
    - *V*
    - *OV*
  - **Definicja**
    - Gdy bity znaku obu argumentów są identyczne a znak wyniku przeciwny do nich, wtedy *O*=1, a w przeciwnym razie *O*=0
  - **Dla operacji ADC Rd, Rr (zgodnie z: AVR Instruction Set)**
    - $O = Rd_7 \bullet Rr_7 \bullet !R_7 + !Rd_7 \bullet !Rr_7 \bullet R_7$

    Gdzie:
    - • - operacja AND
    - Rd_7 i Rr_7 - najstarsze bity argumentów
    - R_7 - najstarszy bit wyniku

- **Flaga *O* (nadmiar), cd.**
  - **Przykład dla operacji na danych 7 bitowych ze znakiem:**
    - **(-53) + (74) = (21)   [0xCB + 0x4A = 0x15]**
      - Mamy przeniesienia z sumowań na pozycjach 6 i 7 wiec *O*=0 oraz mimo że *C*=1 (ignorowane) wynik OK.
    - **(-53) + (-74) = (-127)   [0xCB + 0xB6 = 0x81]**
      - Mamy przeniesienia z sumowań na pozycjach 6 i 7 wiec *O*=0 oraz mimo że *C*=1 (ignorowane) wynik OK.
    - **(+54) + (74) = (128)   [0x36 + 0x4A = 0x80]**
      - Mamy przeniesienie z pozycji 6 ale brak przeniesienia z pozycji 7 oraz znak wyniku jest różny od znaków składników daje *O*=1 czyli błędny wynik
      - lub zgodnie z dokumentacją dla AVR: Rd_7=0 i Rr_7=0 a R_7=1 stąd *O* =1

Programowanie CPU

Systemy Komputerowe

- **Znamy listę instrukcji CPU - ale czy trzeba programować w asemblerze?**
  - **Jak dzisiaj tworzy się oprogramowanie współpracujące ze sprzętem?**
    - **Zasadniczo w C i coraz częściej w C++**
    - **Po co asembler?**
      - **Podstawa dla twórców kompilatorów**
      - **Gdy coś w kodzie działa nie tak**
      - **Gdy fragmenty kodu wygenerowanego z C/C++ działają za wolno**
      - **Gdy kod ma mieścić się w bardzo małych pamięciach**
        - **zagadnienie o źródłach ekonomicznych, MCU z wyliczeń finansowych ma kosztować mniej niż 1% kosztów produkcji całego urządzenia (zabawki, gadżety, ...)**
      - **Gdy chcemy sprawić aby nikt nie był wstanie zrozumieć kodu (;->>)**

Systemy Komputerowe

- **Jak wygląda proces kompilacji**
  - **1 - preprocesing**

gcc -E main.c > main.E

```
main.c:
#include <stdio.h>
int main(int argc, char *argv[]){
    printf("Hello world!\n");
    return 0;
}
```

```
main.E:
# 1 "main.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 31 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 32 "<command-line>" 2
# 1 "main.c"
# 1 "/usr/include/stdio.h" 1 3 4
# 27 "/usr/include/stdio.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/bits/libc-header-start.h" 1 3 4
...
# 2 "main.c" 2
# 2 "main.c"
int main(int argc, char *argv[]){
    printf("Hello world!\n");
    return 0;
}
```

Systemy Komputerowe

- **Jak wygląda proces kompilacji**
  - **2 - właściwa kompilacja C do ASM**

gcc -save-temps main.c

```
main.c:
#include <stdio.h>
int main(int argc, char *argv[]){
    printf("Hello world!\n");
    return 0;
}
```

```
main.s:
 .file   "main.c"
        .text
        .section        .rodata
.LC0:
        .string "Hello world!"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    %edi, -4(%rbp)
        movq    %rsi, -16(%rbp)
        leaq    .LC0(%rip), %rdi
        call    puts@PLT`
        movl    $0, %eax
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (Debian 8.3.0-6) 8.3.0"
        .section        .note.GNU-stack,"",@progbits
```

Systemy Komputerowe

- **Jak wygląda proces kompilacji**
  - **3 - właściwa kompilacja z ASM/C do OBJ (postać pośrednia)**

gcc -c main.c -o main.o
objdump -S main.o

```
main.c:
#include <stdio.h>
int main(int argc, char *argv[]){
    printf("Hello world!\n");
    return 0;
}
```

```
main.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <main>:
   0:   55                      push   %rbp
   1:   48 89 e5                mov    %rsp,%rbp
   4:   48 83 ec 10             sub    $0x10,%rsp
   8:   89 7d fc                mov    %edi,-0x4(%rbp)
   b:   48 89 75 f0             mov    %rsi,-0x10(%rbp)
   f:   48 8d 3d 00 00 00 00    lea    0x0(%rip),%rdi        # 16 <main+0x16>
  16:   e8 00 00 00 00          callq  1b <main+0x1b>
  1b:   b8 00 00 00 00          mov    $0x0,%eax
  20:   c9                      leaveq
  21:   c3                      retq
```

≡ call tyle, że dla jest to instrukcja traktowana jako 64 bitowa (np.: rip zamiast ip)

Adres symboliczny

W tej fazie adres nie znany

Systemy Komputerowe

- **Jak wygląda proces kompilacji**
  - **4 - linkowanie**

```
main.c:
#include <stdio.h>
int main(int argc, char *argv[]){
    printf("Hello world!\n");
    return 0;
}
```

gcc main.c -o main
objdump -DxS main

```
main:     file format elf64-x86-64
main
architecture: i386:x86-64, flags 0x00000150:
HAS_SYMS, DYNAMIC, D_PAGED
start address 0x0000000000001050
Program Header:
...
Dynamic Section:
  NEEDED               libc.so.6
  INIT                 0x0000000000001000
...
Sections:
Idx Name          Size      VMA               LMA               File off  Algn
 ...
 13 .text         00000171  0000000000001050  0000000000001050  00001050  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 ...
 23 .data         00000010  0000000000004020  0000000000004020  00003020  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 24 .bss          00000008  0000000000004030  0000000000004030  00003030  2**0
                  ALLOC
 ...
0000000000001135 <main>:
    1135:      55                      push   %rbp
    ...
    1144:      48 8d 3d b9 0e 00 00    lea    0xeb9(%rip),%rdi     # 2004 <_IO_stdin_used+0x4>
    114b:      e8 e0 fe ff ff          callq  1030 <puts@plt>
...
```

Virtual Memory Adress

Load Memory Adress

Teraz call ma finalny adres

Systemy Komputerowe

- **Jak wygląda proces kompilacji, cd.**
  - **Czy kompilator to czarna skrzynka?**
    - dla kompilatorów otwartych można przejrzeć ich źródła - trudne zdanie!
  - **Jakie informacje o budowie kompilatora GCC można odkryć**
    - jak dowiedzieć się gdzie kompilator GCC zagłada podczas swojej pracy

gcc --print-search-dirs

```
install: /usr/lib/gcc/x86_64-linux-gnu/4.9/

programs: =/usr/lib/gcc/x86_64-linux-gnu/4.9/:/usr/lib/gcc/x86_64-linux-
   gnu/4.9/:/usr/lib/gcc/x86_64-linux-gnu/:/usr/lib/gcc/x86_64-linux-
   gnu/4.9/:/usr/lib/gcc/x86_64-linux-gnu/:/usr/lib/gcc/x86_64-linux-
   gnu/4.9/../../../../x86_64-linux-gnu/bin/x86_64-linux-gnu/4.9/:/usr/lib/gcc/x86_64-
   linux-gnu/4.9/../../../../x86_64-linux-gnu/bin/x86_64-linux-gnu/:/usr/lib/gcc/x86_64-
   linux-gnu/4.9/../../../../x86_64-linux-gnu/bin/

libraries: =/usr/lib/gcc/x86_64-linux-gnu/4.9/:/usr/lib/gcc/x86_64-linux-
   gnu/4.9/../../../../x86_64-linux-gnu/lib/x86_64-linux-gnu/4.9/:/usr/lib/gcc/x86_64-
   linux-gnu/4.9/../../../../x86_64-linux-gnu/lib/x86_64-linux-gnu/:/usr/lib/gcc/x86_64-
   linux-gnu/4.9/../../../../x86_64-linux-gnu/lib/../lib/:/usr/lib/gcc/x86_64-linux-
   gnu/4.9/../../../x86_64-linux-gnu/4.9/:/usr/lib/gcc/x86_64-linux-
   gnu/4.9/../../../x86_64-linux-gnu/:/usr/lib/gcc/x86_64-linux-
   gnu/4.9/../../../../lib/:/lib/x86_64-linux-gnu/4.9/:/lib/x86_64-linux-
   gnu/:/lib/../lib/:/usr/lib/x86_64-linux-gnu/4.9/:/usr/lib/x86_64-linux-
   gnu/:/usr/lib/../lib/:/usr/lib/gcc/x86_64-linux-gnu/4.9/../../../../x86_64-linux-
   gnu/lib/:/usr/lib/gcc/x86_64-linux-gnu/4.9/../../../:/lib/:/usr/lib/
```

Systemy Komputerowe

- **Jak wygląda proces kompilacji, cd.**
  - **Jakie informacje o budowie kompilatora GCC można odkryć**
    - jak dowiedzieć się z jakich plików bibliotecznych korzysta GCC podczas kompilacji i linkowania

gcc -Wl,-t

```
/usr/bin/ld: mode elf_x86_64
/usr/lib/gcc/x86_64-linux-gnu/4.9/../../../x86_64-linux-gnu/crt1.o
/usr/lib/gcc/x86_64-linux-gnu/4.9/../../../x86_64-linux-gnu/crti.o
/usr/lib/gcc/x86_64-linux-gnu/4.9/crtbegin.o
-lgcc_s (/usr/lib/gcc/x86_64-linux-gnu/4.9/libgcc_s.so)
/lib/x86_64-linux-gnu/libc.so.6
(/usr/lib/x86_64-linux-gnu/libc_nonshared.a)elf-init.oS
/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
-lgcc_s (/usr/lib/gcc/x86_64-linux-gnu/4.9/libgcc_s.so)
/usr/lib/gcc/x86_64-linux-gnu/4.9/crtend.o
/usr/lib/gcc/x86_64-linux-gnu/4.9/../../../x86_64-linux-gnu/crtn.o
```

Systemy Komputerowe

- **Jak wygląda proces kompilacji, cd.**
  - **Jakie informacje o budowie kompilatora GCC można odkryć**
    - z jakich makr wbudowanych korzysta GCC:

gcc -E -dM - < /dev/null

```
...
#define __x86_64 1
#define __linux 1
#define __VERSION__ "4.9.2"
#define __amd64__ 1
...
#define __INT8_MAX__ 127
#define __UINT8_MAX__ 255
#define __INT16_MAX__ 32767
#define __UINT16_MAX__ 65535
#define __UINT32_MAX__ 4294967295U
#define __INT64_MAX__ 9223372036854775807L
#define __UINT64_MAX__ 18446744073709551615UL
#define __INT_MAX__ 2147483647
#define __LONG_MAX__ 9223372036854775807L
#define __LONG_LONG_MAX__ 9223372036854775807LL
...
#define __SIZEOF_SHORT__ 2
#define __SIZEOF_INT__ 4
```

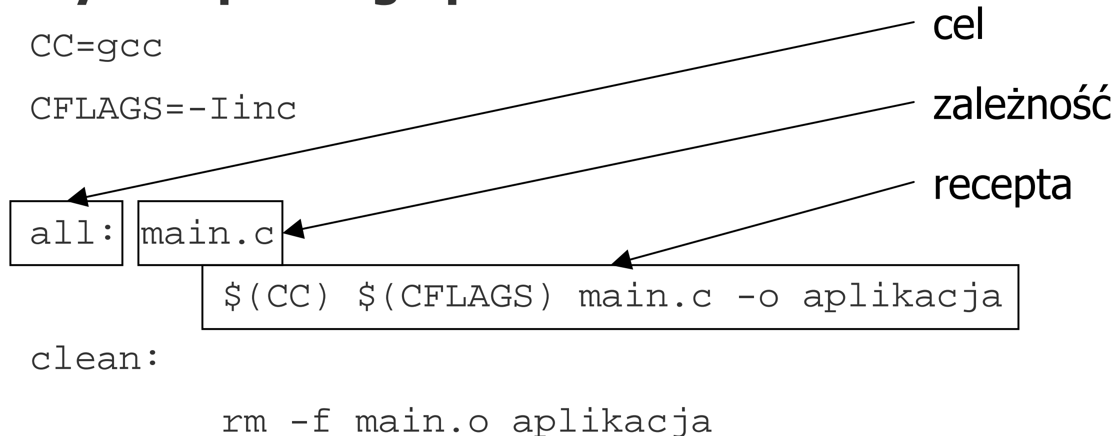
```
#define __SIZEOF_POINTER__ 8
#define __SIZEOF_LONG__ 8
#define __SIZEOF_LONG_LONG__ 8
#define __SIZEOF_FLOAT__ 4
#define __SIZEOF_DOUBLE__ 8
#define __SIZEOF_LONG_DOUBLE__ 16
...
#define __SIZEOF_SIZE_T__ 8
...
#define __INT8_TYPE__ signed char
#define __UINT8_TYPE__ unsigned char
#define __INT16_TYPE__ short int
#define __UINT16_TYPE__ short unsigned int
#define __INT32_TYPE__ int
#define __UINT32_TYPE__ unsigned int
#define __UINT64_TYPE__ long int
#define __UINT64_TYPE__ long unsigned int
...
```

Systemy Komputerowe

- **Jak wygląda proces kompilacji, cd.**
  - **Jak profesjonalnie tworzyć kod z wykorzystaniem GCC i innych narzędzi GNU**
    - Istnieje mnóstwo pakietów IDE (Integrated Development Environment)
      - Eclipse, CodeBlocks, Visual Studio, ...
  - **W każdym przypadku zaleca się automatyzacje procesu kompilacji**
    - Istniej zestaw narzędzi ułatwiających tworzenie plików wynikowych
      - Make, CMake, Autoconf, Automake, Ant(java), ...
  - **Jak działa Make**
    - narzędzie szuka w katalogu wywołania plików Makefile, ...
    - następnie dokonuje kompilacje zgodnie z zapisanymi w tym pliku receptami

Systemy Komputerowe

- **Jak wygląda proces kompilacji, cd.**
  - **Przykład prostego pliku Makefile**




```
clean:
        rm -f main.o aplikacja
```

  - **Bardziej zaawansowany przykład - rekompilacja selektywna**

```
CC=gcc
CFLAGS=-Iinc
all: main.o func.o
        $(CC) main.o func.o -o aplikacja
main.o: main.c
        $(CC) $(CFLAGS) -c main.c -o main.o
func.o: func.c
        $(CC) $(CFLAGS) -Ifunc_inc -c func.c -o func.o
clean:
        rm -f main.o func.o aplikacja
```

Zmiana któregokolwiek z plików w C wymusi rekompilacje tylko tego pliku!

Systemy Komputerowe

- **Jak wygląda proces kompilacji, cd.**
  - **Jak wygenerować zależności dla pliku źródłowego**
    - **zależności do zapisania w pliku Makefile - biblioteki**

gcc -Wl,-y,printf main.c

```
/usr/bin/ld: /lib/x86_64-linux-gnu/libc.so.6: definition of printf
```

- zatem pozwala śledzić symbole i gdzie są implementowane (jeżeli są to funkcje)

    - **zależności do zapisania w pliku Makefile - pliki nagłówkowe**

gcc -MM main.c

```
main.o: main.c
```

puste bo plik main.c nie ma zależności poza tzw. plikami systemowymi

Systemy Komputerowe

- **Jak wygląda proces kompilacji, cd.**
  - **Jak wygenerować zależności dla pliku źródłowego**
    - **zależności bardziej szczegółowe - pliki nagłówkowe**

gcc -M main.c **lub** **gcc -M -MG main.c**

```
main.o: main.c /usr/include/stdc-predef.h /usr/include/stdio.h \
 /usr/include/x86_64-linux-gnu/bits/libc-header-start.h \
 /usr/include/features.h /usr/include/x86_64-linux-gnu/sys/cdefs.h \
 /usr/include/x86_64-linux-gnu/bits/wordsize.h \
 /usr/include/x86_64-linux-gnu/bits/long-double.h \
 /usr/include/x86_64-linux-gnu/gnu/stubs.h \
 /usr/include/x86_64-linux-gnu/gnu/stubs-64.h \
 /usr/lib/gcc/x86_64-linux-gnu/8/include/stddef.h \
 /usr/lib/gcc/x86_64-linux-gnu/8/include/stdarg.h \
 /usr/include/x86_64-linux-gnu/bits/types.h \
 /usr/include/x86_64-linux-gnu/bits/typesizes.h \
 /usr/include/x86_64-linux-gnu/bits/types/__fpos_t.h \
 /usr/include/x86_64-linux-gnu/bits/types/__mbstate_t.h \
 /usr/include/x86_64-linux-gnu/bits/types/__fpos64_t.h \
 /usr/include/x86_64-linux-gnu/bits/types/__FILE.h \
 /usr/include/x86_64-linux-gnu/bits/types/FILE.h \
 /usr/include/x86_64-linux-gnu/bits/types/struct_FILE.h \
 /usr/include/x86_64-linux-gnu/bits/stdio_lim.h \
 /usr/include/x86_64-linux-gnu/bits/sys_errlist.h
```

Sztuczki i operacje w kodzie

Systemy Komputerowe

- „HOW TO" - dla wartości 8 bitowych bez znaku
  - **Jak ustawić określony bit (np.: 1)**

```
uint8_t x;
...
x = x | 0x02;
```

  - **Jak skasować określony bit (np.: 1)**

```
uint8_t x;
...
x = x & 0xFD;          lub    x = x & ~(0x02);
```

  - **Jak zmienić na przeciwny stan określonego bitu (np.: bitu na pozycji 1)**

```
uint8_t i,t;
...
t = i   & ~(0x02); zapamiętanie reszty bitów
i = i   &   0x02;  "wyłowienie bitu zainteresowania"
i = (~i)&   0x02;  negacja bitu „zainteresowania"
i = t   |   i;     nałożenie obu wartości
```

Lub prościej:

```
uint8_t i;
...
i = i ^ 0x02;
```

Systemy Komputerowe

- **Operacje bitowe - „HOW TO”, cd.**
  - **Zbadanie parzystości zmiennej:**

```
uint8_t x;
...
if((x & 0x01)!=0){
        //x nie parzyste
}else{
        //x parzyste
}
```

  - **Przekazanie wybranych trzech bitów (np.: 4,3,2) jako argumentów wywołania zadanej funkcji**

```
uint8_t i;
...
func((i & 0x1C)>>2);   //0x1C to: 0001 1100 binarnie
                       //         czyli ustawione bity: 4,3,2
```

Systemy Komputerowe

- **Łączenie kodu w asemblerze z kodem C**
  - **Najczęściej łączy się C z asemblerem będącym tzw. wstawką, np.:**

```
#define BIT(n) PORTD=clrClkAndData;\
asm __volatile__ (                  \
    "sbrc %2," #n                   \
    "sbi 18,3"                      \
    "sbi 18,5"                      \
    "sbic 16,2"                     \
    "ori %0,1<<" #n                 \
    : "=d" (spiIn) : "0" (spiIn),   \
      "r" (spiOut))

uint8_t spi(uint8_t spiOut) {
    uint8_t spiIn = 0;
    uint8_t clrClkAndData;
          BIT(7);
          BIT(6);
          ...
          BIT(0);
    return spiIn;
}
```

Po pre-procesingu, kompilacji i linkowaniu:

```
//BIT(7)
ldi      r30, 0x32
ldi      r31, 0x00
ldd      r24, Y+1
st       Z, r24
ldd      r24, Y+2
ldd      r25, Y+3
sbrc     r25, 7
sbi      0x12, 3    ;0x12=18
sbi      0x12, 5    ;0x12=18
sbic     0x10, 2    ;0x10=16
ori      r24, 0x80 ;0x80=1<<7
std      Y+2, r24
//BIT(6)
ldi      r30, 0x32
ldi      r31, 0x00
ldd      r24, Y+1
st       Z, r24
ldd      r24, Y+2
ldd      r25, Y+3
sbrc     r25, 6
sbi      0x12, 3
sbi      0x12, 5
sbic     0x10, 2
ori      r24, 0x40 ;0x40=1<<6
std      Y+2, r24
```

Dziękuje za uwagę