x86 assembler and inline assembler in GCC

Michal Sojka
sojkam1@fel.cvut.cz
ČVUT, FEL

Useful instructions

- mov – moves data between registers and memory
  - `mov $1,%eax   # move 1 to register eax`
  - `n: .int 123   # label n points to an integer`
    `# variable`
    `mov n,%eax    # move value of the variable to eax`
  - `mov %eax,%ebx # copy the value in eax to ebx`
- push/pop – stack manipulation
  - Useful when we need to store data for later and we cannot use registers for that
  - `push %eax    # push content of eax to the stack`
  - `pop %ebx     # pop a value from the stack to ebx`

Useful instructions 2

- add – adds two operands
  - add $2,%eax # eax = eax + 2
  - add %eax,%ebx # ebx = ebx + eax
- sub – subtracts two operands
  - sub $2,%eax # eax = eax – 2

Useful instructions 3

- call – calls a subroutine
- ret – returns from a subroutine to the caller

```
plusone:
     add $1,%eax
     ret
main:
      mov $12,%eax
      call plusone
      ...
```

Useful instructions 4

- div – integer division (not a simple instruction)
  http://x86.renejeschke.de/html/file_module_x86_id_72.html
  - 8 bit operand: ax divided by the operand
    result: al = ax / operand, ah = ax % operand
    - mov $42,%ax
      mov $12,%bl
      div %bl # al = 42/12 = 3
  - 16 bit operand: dx:ax divided by the operand
    result: ax = dx:ax / operand, dx = dx:ax % operand
    - mov $0x1,%dx
      mov $0x2345,%ax
      mov $10,%bx
      div %bx # ax = 0x12345 / 10
  - ...

Useful instructions 5

- cmp – compare two values
  - cmp $2,%eax # compare eax with 2 and set eflags register
  - je label # jump to the label if eax was **equal** to 2
  - jl label # jump if eax was **less**
  - jg label # jump if eax was **greater**
  - jlelabel # jump if **less or equal**
  - jge label # jump if **greater or equal**
- Example**:**
  -
```
    cmp $0x30,%al
    jl nodigit
    cmp $0x39,%al
    jg nodigit
digit:
    ... do something ...
nodigit:
    ... handle error
```

Extended assembler

- Dovoluje použití výrazů jazyka C v instrukcích assembleru
- Programátor specifikuje “šablonu assemblerovských instrukcí” (viz `"mov %%esp,%0;" níže`)
- Překladač v šabloně nahradí parametry (např. %0 níže) reálnými operandy (registry, adresami v paměti, ...) na základě specifikovaných omezení
- Kompilátor se nesnaží porozumět tomu, co programátor do šablony napsal.
- Proto musí programátor říct kompilátoru, jak dané instrukce mění stav programu.

```
// Compile with gcc -m32 -O2 -Wall ...
#include <stdio.h>
int main()
{
    void *stack_ptr;
    asm volatile ("mov %%esp,%0;" : "=g" (stack_ptr));
    printf("Value of ESP register is %p\n", stack_ptr);
    return 0;
}
```

Syntaxe extended assembleru

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  int result, op1 = 4, op2 = 2;
  asm volatile (
      "mov %1,%0;"
      "add %2,%0;"
      : "=r" (result)
      : "r" (op1), "r" (op2)
      : "cc");
  printf("result = %d\n", result);
  return 0;

}
```

Syntaxe extended assembleru:

```
asm ( šablona instrukcí
      : výstupní operandy   /* nepovinné */
      : vstupní operandy    /* nepovinné */
      : co je instrukcemi změněno /* nepov. */
    );
```

Syntaxe operandů za ":" je:

```
  <omezení> (<výraz v C>), ...
```

každý operand se vztahuje k odpovídajícímu parametru (%<číslo>) v šabloně (vlevo %0 odpovídá result, %1 odpovídá op1 a %2 odpovídá op2).

https://gcc.gnu.org/onlinedocs/gcc/Extended-Asm.html

Compiles into (see objdump -d ...):

```
...
80482c0: ba 02 00 00 00   mov $0x2,%edx
80482c5: b8 04 00 00 00   mov $0x4,%eax
80482ca: 89 c0            mov %eax,%eax
80482cc: 01 d0            add %edx,%eax
...
```

Omezení v extended assembleru

- V seznamu vstupních nebo výstupních operandů říkají překladači, jaké registry nebo jiné operandy může použít s danou instrukcí v šabloně
  - https://gcc.gnu.org/onlinedocs/gcc/Constraints.html
  - Obecné
    - “r” – any register
      - asm (“mov %0,%eax” : : “r” (var) : “a”) se může přeložit jako “mov %ebx,%eax” s tím, že překladač zajistí, že v registru ebx bode hodnota proměnné var.
    - “m” – memory location
      - asm (“mov %0,%eax” : : “m” (var) : “a”) se přeloží jako “mov ($var),%eax” (v instrukci bude přímo adresa proměnné var).
    - “i” – immediate operand
      - asm (“mov %0,%eax” : : “i” (123) : “a”) se přeloží “mov $123,%eax”
  - Architekturně (HW) závislé
    - “a” – *ax register (for x86)
    - “b” – *bx register (for x86)
  - Modifikátory:
    - “=” před omezením říká, že se jedná o výstupní operand.

Omezení v extended assembleru

- Za 3. dvojtečkou (co je změněno) říkají překladači, co instrukce mění.
- Může to být např.
  - konkrétní registry “a”, “b”, atd.
    - To je nutné jen pokud nejsou uvedeny jako vstup nebo výstup (za 1. nebo 2. dvojtečkou)
  - “cc” – říká, že instrukce modifikují registr příznaků (condition codes, někdy také označovaný jako flags).
    - Toto je potřeba uvést, například pokud šablona obsahuje instrukci add. Ta totiž nastavuje příznak “carry” podle toho, jestli výsledek sčítání přetekl nebo ne.
  - “memory” – říká, že instrukce čtou nebo zapisují do paměti (na jiná místa než ta uvedená jako vstup nebo výstup)
    - Pokud toto neuvedete a nemáte nebo v navazujícím kódu nepoužijete výstupní operand (omezení s “=”), překladač si může myslet, že daný kus assembleru nedělá nic užitečného a “vyoptimalizuje” (smaže) ho.
    - Toto budete potřebovat minimálně pro systémová volání read a write, která pracují s pamětí předanou jádru pomocí ukazatele.