Семинар 5

Ассемблер и память. Продолжение ARM

ARM

- Инструкция
- Статья на хабре
- Как запустить программу под ARM?
  A. купить мак на M1 или M2
  B. Использовать кросс-компилятор и эмулятор

Кросс-компиляция

- Скачиваем компилятор gcc и sysroot:
- Распаковываем:
  - **$ tar -xf *name*.tar.xz**
- В ~/.bashrc прописываем:
  PATH="$PATH:/home/…/gcc-linaro/bin"
  export **LINARO_SYSROOT**="/home/…/sysroot-glibc-linaro"
- Компилируем файл:
  - **aarch64-linux-gnu-gcc**
- Устанавливаем **qemu:**
  - **$ sudo apt-get install qemu-user-static**
- Запускаем бинарь:
  - **$qemu-aarch64-static -L $LINARO_SYSROOT a.out**

Все сразу

- $ mkdir cross_compile; cd cross_compile
- $ wget https://releases.linaro.org/components/toolchain/binaries/7.5-2019.12/aarch64-linux-gnu/gcc-linaro-7.5.0-2019.12-x86_64_aarch64-linux-gnu.tar.xz
- $ wget https://releases.linaro.org/components/toolchain/binaries/7.5-2019.12/aarch64-linux-gnu/sysroot-glibc-linaro-2.25-2019.12-aarch64-linux-gnu.tar.xz
- $ tar -xf sysroot-glibc-linaro-2.25-2019.12-aarch64-linux-gnu.tar.xz
- $ tar –xf gcc-linaro-7.5.0-2019.12-x86_64_aarch64-linux-gnu.tar.xz
- $ vim ~/.bashrc
  - export ARM_GCC=~/cross_compile/gcc-linaro-7.5.0-2019.12-x86_64_aarch64-linux-gnu/bin
  - export LINARO_SYSROOT=~/cross_compile/sysroot-glibc-linaro-2.25-2019.12-aarch64-linux-gnu
  - alias agcc=‘${ARM_GCC}/aarch64-linux-gnu-gcc'
  - alias agdb=‘${ARM_GCC}/aarch64-linux-gnu-gdb'
  - alias aobjdump=‘${ARM_GCC}/aarch64-linux-gnu-objdump'
  - alias arun='qemu-aarch64 -L ${LINARO_SYSROOT}’
- $ source ~/.bashrc
- $ agcc main.S
- $ arun a.out

А если мак?

- Тим Кук о вас позаботится!

```
┌─dragunk@dragunk-osx ~/Downloads/tmp
└─$ gcc main.c
┌─dragunk@dragunk-osx ~/Downloads/tmp
└─$ file a.out
a.out: Mach-O 64-bit executable arm64
┌─dragunk@dragunk-osx ~/Downloads/tmp
└─$ ./a.out
hello world
┌─dragunk@dragunk-osx ~/Downloads/tmp
└─$ gcc -arch x86_64 main.c
┌─dragunk@dragunk-osx ~/Downloads/tmp
└─$ file a.out
a.out: Mach-O 64-bit executable x86_64
┌─dragunk@dragunk-osx ~/Downloads/tmp
└─$ ./a.out
hello world
```

Отладка

$ agcc –g main.S
$ arun -g 1234 ./a.out

В другом терминале:
$ agdb a.out
(gdb) target remote localhost:1234
(gdb) c

Арифметические операции

```
op rd, rs1, rs2        // rd = rs1 op rs2
add Xd, Xa, Xb         // Xd = Xa + Xb
madd Xd, Xa, Xb        // Xd = Xa + Xb

asr Xd, Xa, Xb         // Xd = Xa >> Xb – арифметический сдвиг вправо
lsr Xd, Xa, Xb         // Xd = Xa >> Xb – логический сдвиг вправо
lsl Xd, Xa, Xb         // Xd = Xa << Xb – логический сдвиг влево

sxtb Xd, Xa            // Xd = Xa[0:7] – знаковое копирование 1 байта
uxth Xd, Xa            // Xd = Xa[0:15] – беззнаковое копирование 2 байт
```

Calling Convention

Регистры	Назначение
`x0` ... `x7`	аргументы функции и возвращаемое значение (`x0`)
`x8` ... `x18`	временные регистры, для которые не гарантируется сохранение результата, если вызывать какую-либо функцию
`x19` ... `x28`	регистры, для которых гарантируется, что вызываемая функция их не будет портить
`x29`	указатель на границу фрейма функции, обычно используется отладчиком
`x30`	адрес возврата из функции
`sp`	указатель на вершину стека

Переходы

b LABEL / b x – безусловный переход (= jmp)
bl LABEL / bl x - переход + link (x30 := адрес возврата)
beq / bne / … - условные переходы

ret – прыжок на адрес из регистра x30

Условия

EQ	equal (Z)
NE	not equal (!Z)
CS or HS	carry set / unsigned higher or same (C)
CC or LO	carry clear / unsigned lower (!C)
MI	minus / negative (N)
PL	plus / positive or zero (!N)
VS	overflow set (V)
VC	overflow clear (!V)
HI	unsigned higher (C && !Z)
LS	unsigned lower or same (!C || Z)
GE	signed greater than or equal (N == V)
LT	signed less than (N != V)
GT	signed greater than (!Z && (N == V))
LE	signed less than or equal (Z || (N != V))

Работа с памятью

ldr Rd, [Xa] – чтение из память по адресу Xa, запись в регистр Rd
str Ra, [Xd] – запись в память по адресу Xd из регистра Ra

ldr / str + [-/s] + [b/h/w] – сохранить / загрузить + (без)/знаковое + ширина
ldrb / strb -> uint8_t
ldrsw / strsw -> int32_t

```
ldr  x0, [x1, 8]        // x0 = *(x1 + 8)
ldr  x0, [x1, x2, lsl 3]  // x0 = *(x1 + x2 * (1 << 3))
ldur x0, [x1, 5]        // x0 = *(x1 + 5)

stp x19, x20, [x8], #16      // stp x19, x20, [x8] + add x8, x8, #16
stp x19, x20, [x8, #16]      // add x8, x8, #16 + stp x19, x20, [x8]
```

Сохранение на стек

```
some_func:
    stp x29, x30, [sp, #16]!
    // ...
    ldp x29, x30, [sp], #0x16
    ret
```

syscalls

arch	syscall NR	return	arg0	arg1	arg2	arg3	arg4	arg5
arm	r7	r0	r0	r1	r2	r3	r4	r5
arm64	x8	x0	x0	x1	x2	x3	x4	x5
x86	eax	eax	ebx	ecx	edx	esi	edi	ebp
x86_64	rax	rax	rdi	rsi	rdx	r10	r8	r9

N.B.! Порядок аргументов и номера сисколов отличаются даже между x86 и x86_64!

- Таблица с аргументами сисколов

x86_64 (64-bit)

Compiled from Linux 4.14.0 headers.

NR	syscall name	references	%rax	arg0 (%rdi)	arg1 (%rsi)	arg2 (%rdx)
0	read	man/ cs/	0x00	unsigned int fd	char *buf	size_t count
1	write	man/ cs/	0x01	unsigned int fd	const char *buf	size_t count
2	open	man/ cs/	0x02	const char *filename	int flags	umode_t mode
3	close	man/ cs/	0x03	unsigned int fd	-	-

x86 (32-bit)

Compiled from Linux 4.14.0 headers.

NR	syscall name	references	%eax	arg0 (%ebx)	arg1 (%ecx)	arg2 (%edx)
0	restart_syscall	man/ cs/	0x00	-	-	-
1	exit	man/ cs/	0x01	int error_code	-	-
2	fork	man/ cs/	0x02	-	-	-
3	read	man/ cs/	0x03	unsigned int fd	char *buf	size_t count
4	write	man/ cs/	0x04	unsigned int fd	const char *buf	size_t count
5	open	man/ cs/	0x05	const char *filename	int flags	umode_t mode
6	close	man/ cs/	0x06	unsigned int fd	-	-

inline asssembly

```
int no = 100, val ;
    asm ("movl %1, %%ebx;"
         "movl %%ebx, %0;"
         : "=r" ( val )          /* output */
         : "r" ( no )          /* input */
         : "%ebx"          /* clobbered register */
     );
```

- Почитать [тут]

macro

```
.macro CONDITIONAL_ADD, cond, dest, src1, src2
    .if \cond == 1
        ADD \dest, \src1, \src2
    .else
        SUB \dest, \src1, \src2
    .endif
.endm


.macro add_regs dest, src1, src2=0
    add \dest, \src1, \src2
.endm
```

Базовые макросы