

2023-10-02-IHS-Apresentacao

C Compiler para Arquitetura Poxim

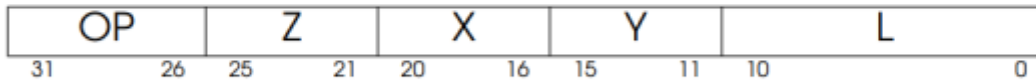
Não existe Hardware Poxim (ainda) para rodar código de máquina nele

E por isso, não Existe implementação de C para essa Arquitetura

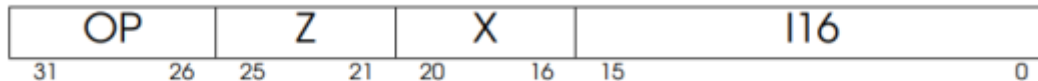
- ▶ Arquitetura Poxim
 - ▶ *Complexity-Reduced Instruction Set Processor* (CRISP)
 - ▶ Didática, hipotética e simples com 32 bits
 - ▶ Memória Von Neumann de 32 KiB
 - ▶ 3 formatos de instruções

OPERATIONS

- ▶ 6 bits para operação (OP)
 - ▶ 5 bits para operandos (Z, X, Y)
 - ▶ 11 bits para uso livre (L)
- ▶ Formato **U**(OP, Z, X, Y, L)



- ▶ 6 bits para operação (OP)
 - ▶ 5 bits para operandos (Z, X)
 - ▶ 16 bits para imediato (I16)
- ▶ Formato **F**(OP, Z, X, I16)



- ▶ 6 bits para operação (OP)
 - ▶ 26 bits para imediato (I26)
- ▶ Formato **S**(OP, I26)



Existe Compilador de C Para AMD64

Ideia: Adaptar para Poxim

x86_64-asm.h
x86_64-gen.c
x86_64-link.c



poxim-asm.c
poxim-gen.c
poxim-link.c

Transforme Isso

```
#include "_start.h"

int fibonnacci(int n) {
    if (n < 2) {
        return n;
    }
    return fibonnacci(n - 1) + fibonnacci(n - 2);
}

int main(void) {
    int a = fibonnacci(6); // = 8
    int b = fibonnacci(5); // = 5
    return a+b; // r2 should have 13, 0xd
}
```

Nisso

```

# Main
1d0: 28 00 01 80      push    r6
1d4: 28 00 01 c0      push    r7
1d8: 08 de 00 00      mov     r6, sp
1dc: 08 e6 00 00      mov     r7, r6
1e0: 10 07 3f 01      sra     r0, r7, r7, 1      # 0x2
1e4: 4f de 00 08      subi   sp, sp, 8
1e8: 04 20 00 06      movs    r1, 6
1ec: 28 00 00 40      push    r1
1f0: e7 ff ff da      call    -38<<2            # 0x15c
1f4: 4b de 00 04      addi    sp, sp, 4
1f8: 74 47 00 00      s32     [r7+0]<<2, r2
1fc: 04 20 00 05      movs    r1, 5
200: 28 00 00 40      push    r1
204: e7 ff ff d5      call    -43<<2            # 0x15c
208: 4b de 00 04      addi    sp, sp, 4
20c: 74 47 ff ff      s32     [r7-1]<<2, r2
210: 68 27 00 00      l32     r1, [r7+0]<<2
214: 68 47 ff ff      l32     r2, [r7-1]<<2
218: 08 21 10 00      add     r1, r1, r2
21c: 08 41 00 00      mov     r2, r1
220: 0b c6 00 00      mov     sp, r6
224: 2c 00 01 c0      pop     r7
228: 2c 00 01 80      pop     r6
22c: 7c 00 00 00      ret

```

Interface Direta com Hardware sem Hardware?

Como sabemos que estamos compilando corretamente?

Poxim Dump e Poxim Interprete

.bin

```
04 20 00 05
28 00 00 40
e7 ff ff d5
4b de 00 04
74 47 ff ff
68 27 00 00
68 47 ff ff
08 21 10 00
08 41 00 00
0b c6 00 00
2c 00 01 c0
2c 00 01 80
7c 00 00 00
```

poxim-dump



.dump

```
movs    r1, 5
push    r1
call    -43<<2      # 0x15c
addi    sp, sp, 4
s32     [r7-1]<<2, r2
l32     r1, [r7+0]<<2
l32     r2, [r7-1]<<2
add     r1, r1, r2
mov     r2, r1
mov     sp, r6
pop     r7
pop     r6
ret
```



poxim-interp

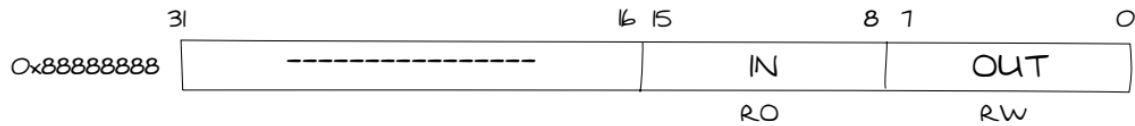
.interp

```
0x00000218: add r1,r1,r2      R1=R1+R2=0x00000000,SR=0x00000000
0x0000021C: add r2,r1,r0      R2=R1+R0=0x00000000,SR=0x00000000
0x00000220: add sp,r6,r0      SP=R6+R0=0x00007FE8,SR=0x00000000
0x00000224: pop r7            {R7}=MEM[0x00007FEC]{0x00001FFD}
0x00000228: pop r6            {R6}=MEM[0x00007FF0]{0x00007FF4}
0x0000022C: ret              PC=MEM[0x00007FF4]=0x00000028
0x00000028: call -9          PC=0x00000008, MEM[0x00007FF4]=0x0000002C
0x00000008: int 0            CR=0x00000000, PC=0x00000000
[END OF SIMULATION]
```

Bugs podem acontecer em qualquer uma dessas etapas
(confie em mim, eu sei)

Visualização no Terminal 0x88888888

- ▶ Registrador da interface serial de texto (Terminal)
 - ▶ Não suporta interrupção
 - ▶ Escrita e leitura sequencial de bytes



```

0x00002580: pop r6
0x00002590: ret
0x00000028: call -9
0x00000008: int 0
[TERMINAL]

-----
>> boolean_and_arithmetic_play <<

1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz

```

Conhecimentos Necessário Para:

Compilação no Formato Elf

Gerar código de Máquina para uma arquitetura Hipotetica

Linkar: Ajeitar as chamadas de funções diretas e indiretas; Ajeitar endereços das sections da rodata e data

e Rodar ?

SER BOM E LER DOCUMENTAÇÃO

EXEMPLO DE DOCUMENTAÇÃO:

```
(v == VT_CONST) {  
    /* XXX handle globals */  
    if (fc ≥ -1 && fc ≤ 8) {  
        out_op(IL_OP_LDC_I4_M1 + fc + 1);  
    } else {  
        out_opi(IL_OP_LDC_I4, fc);  
    }  
}
```

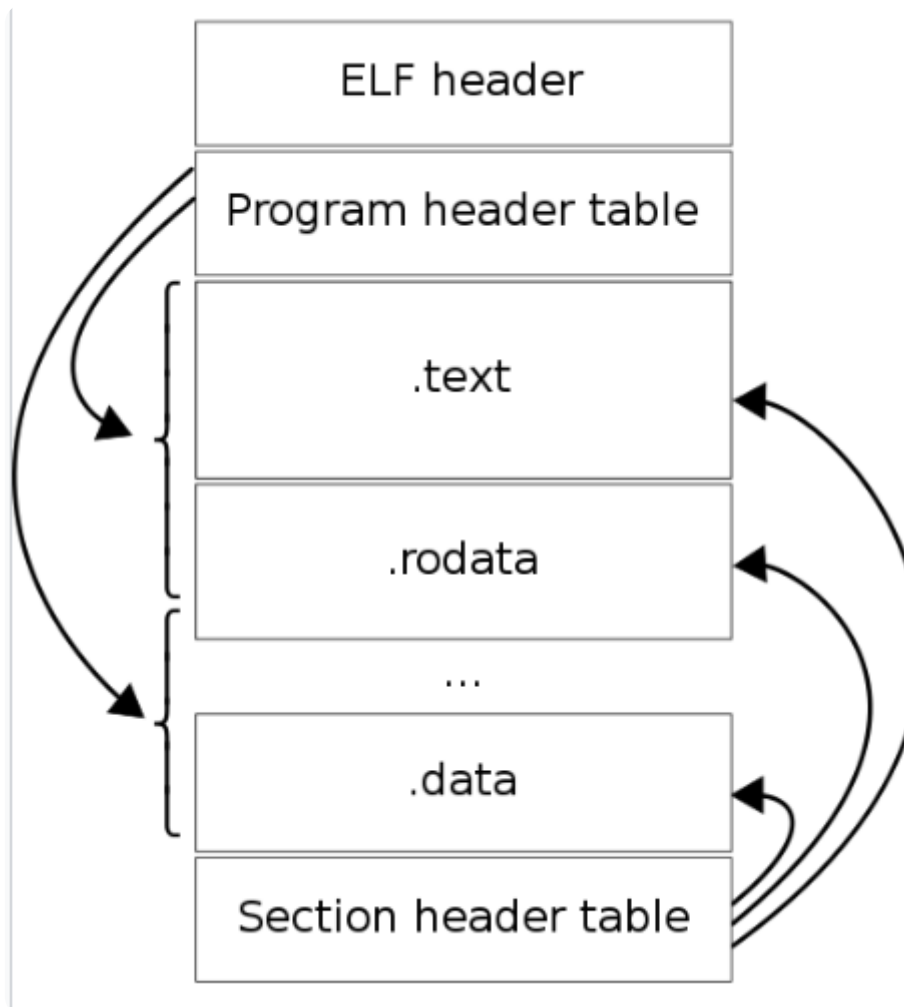
```
/* don't record fields or anonymous symbols */  
/* XXX simplify */  
if (!(v & SYM_FIELD) && (v & ~SYM_STRUCT) < SYM_FIRST_ANOM) {  
    /* record symbol in token array */  
    ts = table_ident[(v & ~SYM_STRUCT) - TOK_IDENT];  
}
```

```
val = vtop->c.i;  
/* XXX make code faster ? */  
if ((vtop->r & (VT_SYM | VT_CONST)) == (VT_SYM | VT_CONST) &&  
    vtop->sym->v ≥ SYM_FIRST_ANOM &&
```

```
1 /* *****/  
2 /* global variables */  
3  
4 /* XXX get rid of this ASAP (or maybe not) */  
5
```

```
6 } else {  
7     /* XXX do it */  
8 }  
9
```

ELF



Ganho o Direito de definir uma ABI para Poxim.

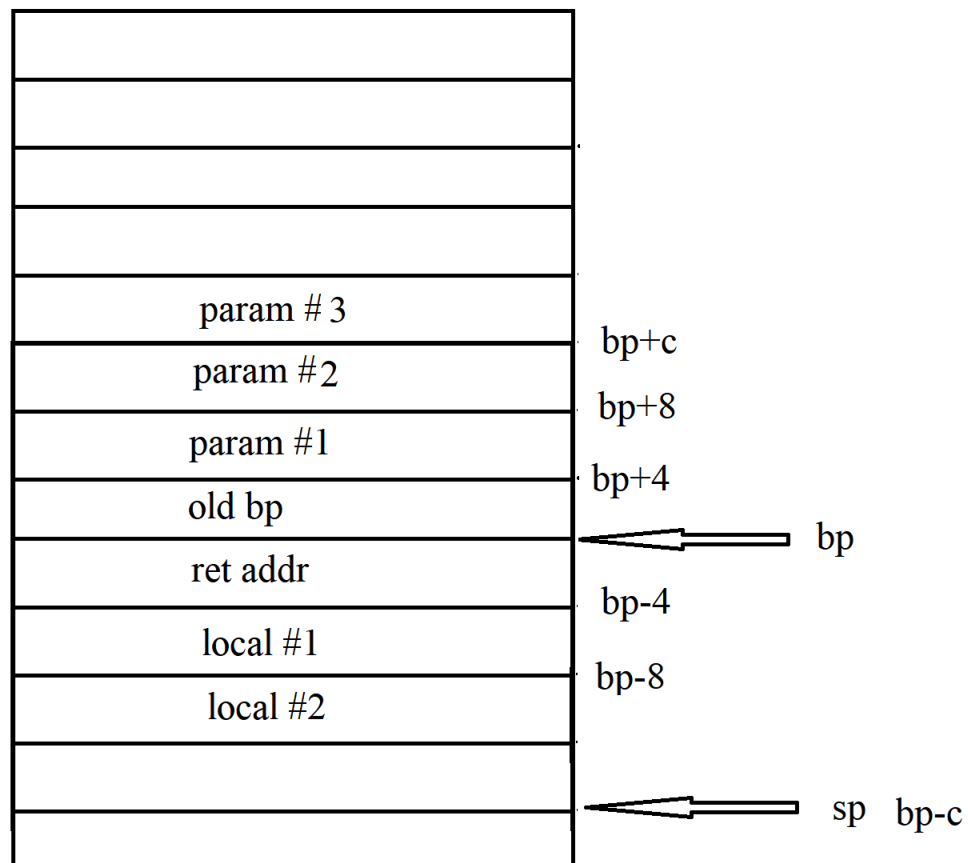
Variáveis Locais definidos no stack.

Passagem de parâmetros pelo stack tbm.

Fácil D+

(não é)

0xff7c

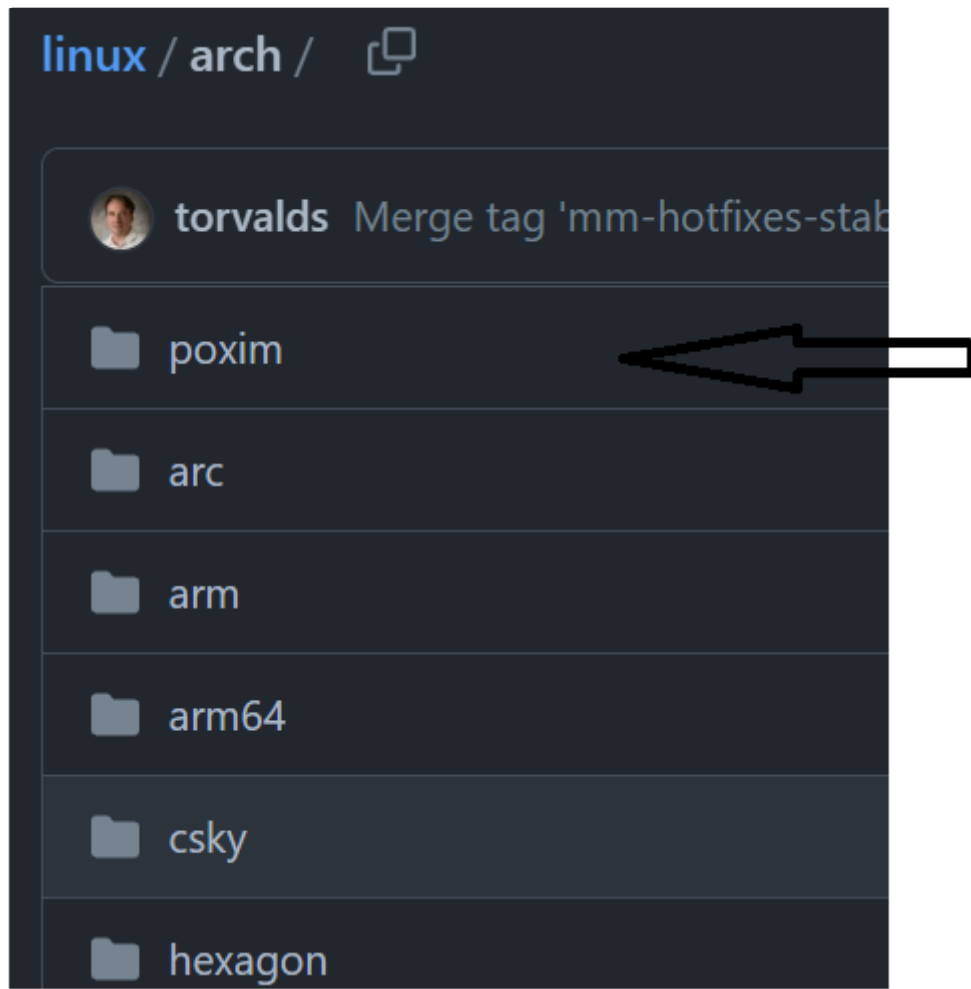


0x0000

Registradores Salvos *sp r6 r7* e cabou.

Próximos Passos

Com um compilador C, o que impede adaptar Linux para esse processador ? Driver Para Poxim hmm.



Agora apresentar o exemplo com a maioria das coisas que funcionam;.,;