The general format of an if statement is

```
if(condition)
{
  do-stuff-here
}
else if(condition) //this is an optional condition
{
  do-stuff-here
}
Else
{
  do-stuff-here
}
```

If statements use 3 important instructions in assembly:

- *cmpq source2, source1: é como calcular a-b sem definir o destino*
- *testq source2, source1: é como calcular a&b sem definir o destino*

As instruções de salto são usadas para transferir o controle para instruções diferentes, e há diferentes tipos de saltos:

Jump Type	Description
jmp	Unconditional
je	Equal/Zero
jne	Not Equal/Not Zero
js	Negative
jns	Nonnegative
jg	Greater
jge	Greater or Equal
jl	Less
jle	Less or Equal
ja	Above(unsigned)
jb	Below(unsigned)

Os dois últimos valores da tabela referem-se a inteiros sem sinal. Inteiros sem sinal não podem ser negativos, enquanto inteiros com sinal representam valores positivos e negativos. Como o computador precisa diferenciá-los, ele usa métodos diferentes para interpretar esses valores.

Para inteiros com sinal, ele usa algo chamado representação de complemento de dois e, para inteiros sem sinal, usa cálculos binários normais.

Inicie r2 com
r2 -d if1

Lembre-se de executar
e asm.syntax=att

E execute os seguintes comandos:
aaa
afl
pdf @main

Isso analisa o programa, lista as funções e desmonta a função principal.

```
[0x7f374d371090]> pdf @main
/ (fcn) main 43
|   int main (int argc, char **argv, char **envp);
|           ; var int32_t var_8h @ rbp-0x8
|           ; var int32_t var_4h @ rbp-0x4
|           ; DATA XREF from entry0 (0x55ae5283650d)
|           0x55ae528365fa      55              pushq %rbp
|           0x55ae528365fb      4889e5          movq %rsp, %rbp
|           0x55ae528365fe      c745f8030000.   movl $3, var_8h
|           0x55ae52836605      c745fc040000.   movl $4, var_4h
|           0x55ae5283660c      8b45f8          movl var_8h, %eax
|           0x55ae5283660f      3b45fc          cmpl var_4h, %eax
|       ,=< 0x55ae52836612      7d06            jge 0x55ae5283661a
|       |   0x55ae52836614      8345f805        addl $5, var_8h
|      ,==< 0x55ae52836618      eb04            jmp 0x55ae5283661e
|      |`-> 0x55ae5283661a      8345fc03        addl $3, var_4h
|      |    ; CODE XREF from main (0x55ae52836618)
|      `--> 0x55ae5283661e      b800000000      movl $0, %eax
|           0x55ae52836623      5d              popq %rbp
|           0x55ae52836624      c3              retq
\
```

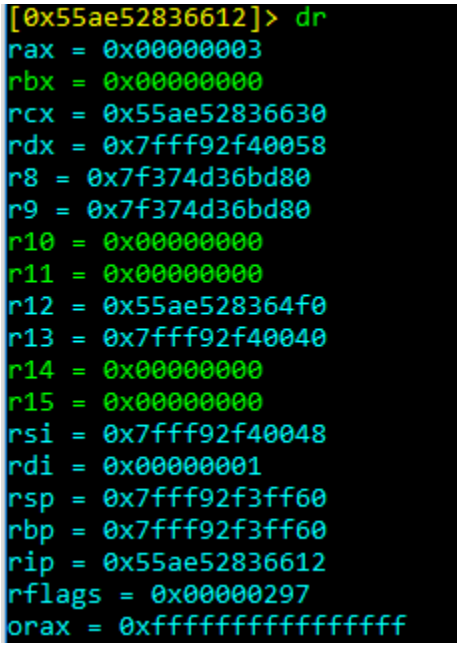
Começaremos então definindo um ponto de interrupção no jge e na instrução jmp usando o comando:
db 0x55ae52836612(which is the hex address of the jge instruction)
db 0x55ae52836618(which is the hex address of the jmp instruction)

Adicionamos pontos de interrupção para interromper a execução do programa nesses pontos, para que possamos ver o estado do programa.
Isso mostrará o seguinte:

Agora, executamos dc para iniciar a execução do programa, que iniciará a execução e parará no ponto de interrupção. Vamos examinar o que aconteceu antes de atingir o ponto de interrupção:

- As primeiras 2 linhas são sobre como empurrar o ponteiro do quadro para o empilhador e salvá-lo (é sobre como as funções são chamadas e será examinado mais tarde)
- As próximas 3 linhas tratam da atribuição dos valores 3 e 4 aos argumentos/variáveis locais var_8h e var_4h. Em seguida, o valor em var_8h é armazenado no registrador %eax.
- A instrução cmpl compara o valor de eax com o do argumento var_8h

Para visualizar o valor dos registros, digite
dr



Podemos ver que o valor de rax, que é a versão de 64 bits de eax, contém 3. Vimos que a instrução jge está saltando com base no fato de o valor de eax ser maior que var_4h. Para ver

o que está em var_4h, podemos ver que, no topo da função principal, ela nos informa a posição de var_4h. Execute o comando:

px @ rbp-0x4

E isso mostra o valor de 4.

Sabemos que eax contém 3 e 3 não é maior que 4, portanto, o salto não será executado. Em vez disso, ele avançará para a próxima instrução. Para verificar isso, execute o comando ds, que busca/avança para a próxima instrução.

```
/ (fcn) main 43
|   int main (int argc, char **argv, char **envp);
|           ; var int32_t var_8h @ rbp-0x8
|           ; var int32_t var_4h @ rbp-0x4
|           ; DATA XREF from entry0 (0x55ae5283650d)
|           0x55ae528365fa      55              pushq %rbp
|           0x55ae528365fb      4889e5          movq %rsp, %rbp
|           0x55ae528365fe      c745f8030000.   movl $3, var_8h
|           0x55ae52836605      c745fc040000.   movl $4, var_4h
|           0x55ae5283660c      8b45f8          movl var_8h, %eax
|           0x55ae5283660f      3b45fc          cmpl var_4h, %eax
|       ,=< 0x55ae52836612 b    7d06            jge 0x55ae5283661a
|       |   ;-- rip:
|       |   0x55ae52836614      8345f805        addl $5, var_8h
|      ,==< 0x55ae52836618 b    eb04            jmp 0x55ae5283661e
|      |`-> 0x55ae5283661a      8345fc03        addl $3, var_4h
|      |    ; CODE XREF from main (0x55ae52836618)
|      `--> 0x55ae5283661e      b800000000      movl $0, %eax
|           0x55ae52836623      5d              popq %rbp
\           0x55ae52836624      c3              retq
[0x55ae52836612]>
```

O rip (que é o ponteiro da instrução atual) indica que ele avança para a próxima instrução — o que demonstra que estamos corretos. A instrução atual então adiciona 5 a var_8h, que é um argumento local. Para verificar se isso realmente acontece, primeiro verifique o valor de var_8h, execute ds e verifique o valor novamente. Isso mostrará que ele incrementa em 5.

```
[0x55ae52836612]> px @rbp-0x8
- offset -      0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x7fff92f3ff58  0300 0000 0400 0000 3066 8352 ae55 0000  ........0f.R.U..
0x7fff92f3ff68  970b fa4c 377f 0000 0100 0000 0000 0000  ...L7...........
0x7fff92f3ff78  4800 f492 ff7f 0000 0080 0000 0100 0000  H...............
0x7fff92f3ff88  fa65 8352 ae55 0000 0000 0000 0000 0000  .e.R.U..........
0x7fff92f3ff98  976f 608c 8d2f efdf f064 8352 ae55 0000  .o`../...d.R.U..
0x7fff92f3ffa8  4000 f492 ff7f 0000 0000 0000 0000 0000  @...............
0x7fff92f3ffb8  0000 0000 0000 0000 976f e0be 6caf 4c8b  .........o..l.L.
0x7fff92f3ffc8  976f 9e56 7f13 dd8a 0000 0000 ff7f 0000  .o.V............
0x7fff92f3ffd8  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x7fff92f3ffe8  3307 384d 377f 0000 3866 364d 377f 0000  3.8M7...8f6M7...
0x7fff92f3fff8  3161 0700 0000 0000 0000 0000 0000 0000  1a..............
0x7fff92f40008  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x7fff92f40018  f064 8352 ae55 0000 4000 f492 ff7f 0000  .d.R.U..@.......
0x7fff92f40028  1a65 8352 ae55 0000 3800 f492 ff7f 0000  .e.R.U..8.......
0x7fff92f40038  1c00 0000 0000 0000 0100 0000 0000 0000  ................
0x7fff92f40048  9417 f492 ff7f 0000 0000 0000 0000 0000  ................
[0x55ae52836612]> ds
[0x55ae52836612]> px @rbp-0x8
- offset -      0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x7fff92f3ff58  0800 0000 0400 0000 3066 8352 ae55 0000  ........0f.R.U..
0x7fff92f3ff68  970b fa4c 377f 0000 0100 0000 0000 0000  ...L7...........
0x7fff92f3ff78  4800 f492 ff7f 0000 0080 0000 0100 0000  H...............
0x7fff92f3ff88  fa65 8352 ae55 0000 0000 0000 0000 0000  .e.R.U..........
0x7fff92f3ff98  976f 608c 8d2f efdf f064 8352 ae55 0000  .o`../...d.R.U..
0x7fff92f3ffa8  4000 f492 ff7f 0000 0000 0000 0000 0000  @...............
0x7fff92f3ffb8  0000 0000 0000 0000 976f e0be 6caf 4c8b  .........o..l.L.
0x7fff92f3ffc8  976f 9e56 7f13 dd8a 0000 0000 ff7f 0000  .o.V............
0x7fff92f3ffd8  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x7fff92f3ffe8  3307 384d 377f 0000 3866 364d 377f 0000  3.8M7...8f6M7...
0x7fff92f3fff8  3161 0700 0000 0000 0000 0000 0000 0000  1a..............
0x7fff92f40008  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x7fff92f40018  f064 8352 ae55 0000 4000 f492 ff7f 0000  .d.R.U..@.......
0x7fff92f40028  1a65 8352 ae55 0000 3800 f492 ff7f 0000  .e.R.U..8.......
0x7fff92f40038  1c00 0000 0000 0000 0100 0000 0000 0000  ................
0x7fff92f40048  9417 f492 ff7f 0000 0000 0000 0000 0000  ................
```

Observe que, como estamos verificando o endereço exato, precisamos verificar apenas o deslocamento 0. O valor armazenado na memória é armazenado em hexadecimal.

A próxima instrução é um salto incondicional e apenas zera o registrador eax. A instrução popq envolve remover um valor da pilha e lê-lo, e a instrução return define esse valor removido para o ponteiro de instrução atual. Nesse caso, indica que a execução do programa foi concluída. Para entender melhor como uma instrução if funciona, você pode verificar o arquivo C correspondente na mesma pasta.

Parte 2: Execução

Identificando que
var_8h vale 8 no início de main
var_4h vale 2 no iníco de main

jle é uma instrução de if que condiz menor ou igual à
Tem-se a seguinte comparação: var_8h (8) > var_4h (2)

Se verdade, addl irá adicionar 1 em var_8h
Se falso, addl irá adicionar 7 em var_4h

Antes do fim de main, os valores das variáveis são
var_8h = 9
var_4h = 2