Logika cyfrowa

Praktyczna lista zadań nr 13

Termin: 9 czerwca 2024 godzina 30:00

Uwaga! Poniższe zadania należy rozwiązać przy użyciu języka SystemVerilog, sprawdzić w DigitalJS oraz wysłać w systemie Web-CAT na SKOS. Należy pamiętać, aby nazwy portów nadesłanego modułu zgadzały się z podanymi w treści zadania. Wysłany plik powinien mieć nazwę `toplevel.sv`. **Nie przestrzeganie tych zasad będzie skutkować przyznaniem 0 punktów.**

1. W zadaniu z listy praktycznej nr 10 należało zaimplementować prosty kalkulator używający odwrotnej notacji polskiej. W tym zadaniu ten kalkulator, po niewielkim rozszerzeniu, stanie się częścią programowalnego kalkulatora, który będzie w stanie obliczać np. potęgi, silnię lub NWD.

   Rozwiązanie powinno składać się z wielu modułów, wśród których muszą znaleźć się moduły opisane poniżej.

   Zmiany w module kalkulatora

   Wejście `op` modułu kalkulatora powinno być teraz trójbitowe. Dodatkowe operacje powinny być następujące:

   - 0 (g, od **g**reater) – ustawia wartość na szczycie stosu na 1, jeśli była większa niż 0, w przeciwnym wypadku ustawia na 0,
   - 4 (s, od **s**wap) – zamiana miejscami dwóch najwyżej położonych wartości na stosie,
   - 5 (l, od **l**oad) – załadowanie na szczyt stosu wartości położonej tyle słów w głąb stosu (nie licząc aktualnego szczytu), ile wynosi wartość na szczycie (wartość 0 oznacza zduplikowanie wartości umieszczonej zaraz pod szczytem),
   - 6 (p, od **p**op) – usunięcie wartości ze szczytu stosu (zmniejsza stos o 1),
   - 7 – ta sama operacja, co powyżej (nr 6).

   Należy też dodać 1-bitowy sygnał `en` – kiedy ten sygnał jest niski, kalkulator powinien nie zmieniać stanu. Nazwę wejścia zegarowego `step` można zmienić na `clk`.

   Są to jedyne wymagane zmiany w tym module. Po ich wykonaniu, moduł powinien zostać użyty w całości, bez dalszych zmian interfejsu, w rozwiązaniu tego zadania.

   Poniższe przykłady ilustrują właściwe działanie tych operacji:

   - 5 g → 1
   - 0 g → 0
   - 5 - g → 0
   - 1 2 s p → 2
   - 3 4 0 l s p s p → 4
   - 3 4 1 l s p s p → 3

   Moduł pamięci kodu

   Pamięć kodu powinna być pamięcią RAM z odczytem asynchronicznym, z słowem 16-bitowym i adresami 10-bitowymi. Dla potrzeb testowania można ją inicjalizować blokiem `initial`, ale sprawdzaczka będzie ładować program przez interfejs zewnętrzny.

   Moduł sterujący

   Moduł ten może być jednocześnie modułem głównym układu. Jego zadaniem jest ładowanie programu do pamięci kodu (w trybie gotowości) oraz odczytywanie instrukcji i przekazywanie ich do modułu kalkulatora (w trybie pracy).

   Moduł powinien mieć następujące wejścia i wyjścia:

   - `clk` – wejście sygnału zegara,
   - `nrst` – zanegowane wejście resetu asynchronicznego (stan niski resetuje),

- `addr` – 10-bitowe wejście adresu interfejsu pamięci,
- `wr` – 1-bitowe wejście zapisu interfejsu pamięci,
- `datain` – 16-bitowe wejście interfejsu pamięci,
- `start` – 1-bitowe wejście uruchamiające obliczenia,
- `ready` – 1-bitowe wyjście sygnalizujące gotowość układu do rozpoczęcia pracy,
- `out` – 16-bitowe wyjście pokazujące wartość na szczycie stosu.

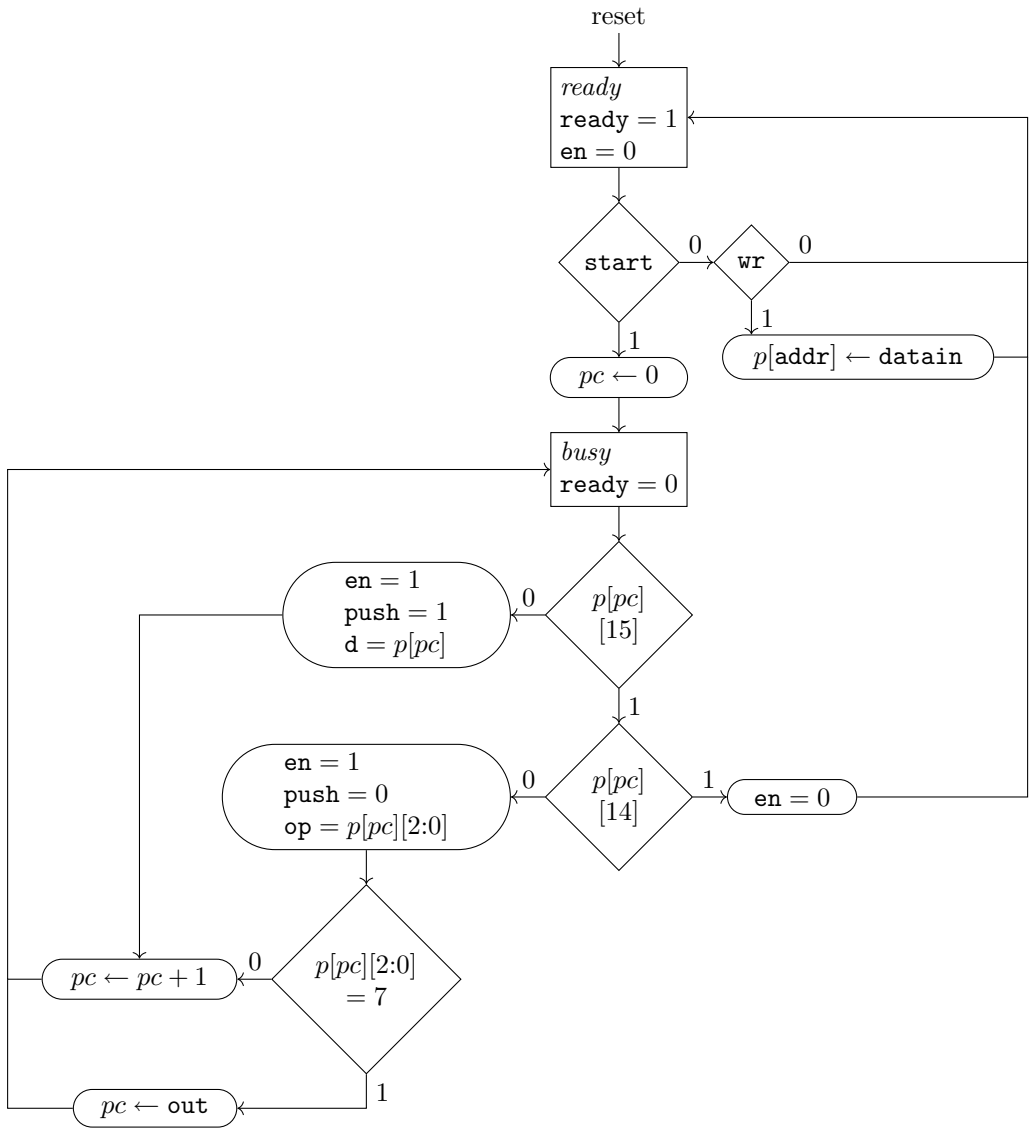
W skład tego modułu powinien wchodzić 10-bitowy rejestr wskaźnika instrukcji `pc`. W momencie rozpoczęcia pracy jego wartość powinna być równa 0.

Słowo z pamięci kodu powinno być interpretowane następująco:

- Jeśli najstarszy bit jest równy 0, należy załadować najmłodsze 15 bitów, interpretowanych jako wartość bez znaku, do kalkulatora.
- Jeśli najstarszy bit jest równy 1, a poprzedni bit jest równy 0, najmłodsze 3 bity należy zinterpretować jako kod operacji dla kalkulatora.

  Jeśli jednocześnie 3 najmłodsze bity są równe 1 (wybrana jest operacja nr 7), należy ustawić wskaźnik instrukcji na wartość na szczycie stosu. Operacja nr 7 jest więc interpretowana jako skok (j, od **j**ump).
- Jeśli dwa najstarsze bity są równe 1, należy zakończyć pracę i przejść w tryb gotowości.

Działanie modułu sterującego można opisać poniższym diagramem algorytmicznym. W diagramie występują zarówno nazwy portów modułu sterującego (np. `ready`), jak i modułu kalkulatora (np. `push`), który jest przez niego sterowany.



Programy przykładowe

Aby umożliwić samodzielne przetestowanie programowalnego kalkulatora, udostępniam poniżej kilka przykładowych programów. Litery x, y oznaczają miejsca, w których należy wpisać stałą – parametr programu. Każdy program jest zapisany w dwóch formach: „czytelnej” (odwrotna notacja polska z dodatkowymi operacjami) i binarnej (zgodnej z opisem podanym wcześniej).

- Obliczanie silni liczby x:
  4 x 6 j 32 j 0 l g 3 * 14 + j 1 28 j 0 l 27 2 l 1 - + 6 j * s p s j

  ```
  0004    x 0006 8007 0020 8007 0000 8005 8000 0003 8003 000e 8002 8007 0001 001c
  8007 0000 8005 001b 0002 8005 0001 8001 8002 0006 8007 8003 8004 8006 8004 8007
  ffff
  ```

- Obliczanie x do potęgi y:
  5 x y 7 j 37 j 0 l g 3 * 15 + j 1 31 j 1 l 30 3 l 3 l 1 - + 7 j * s p s p s j

  ```
  0005    x    y 0007 8007 0025 8007 0000 8005 8000 0003 8003 000f 8002 8007 0001
  001f 8007 0001 8005 001e 0003 8005 0003 8005 0001 8001 8002 0007 8007 8003 8004
  8006 8004 8006 8004 8007 ffff
  ```

- Obliczanie NWD liczb x i y:
  5 x y 7 j 68 j 1 l 1 l - + 0 l - g s g + g 4 * 26 + j 1 l 62 j 1 l 1 l - + g 9 * 42 + j 49 1 l 3 l 7 j 62 j 62 2 l 2 l - + 2 l 7 j s p s p s j

  ```
  0005    x    y 0007 8007 0044 8007 0001 8005 0001 8005 8001 8002 0000 8005 8001
  8000 8004 8000 8002 8000 0004 8003 001a 8002 8007 0001 8005 003e 8007 0001 8005
  0001 8005 8001 8002 8000 0009 8003 002a 8002 8007 0031 0001 8005 0003 8005 0007
  8007 003e 8007 003e 0002 8005 0002 8005 8001 8002 0002 8005 0007 8007 8004 8006
  8004 8006 8004 8007 ffff
  ```

Dla zainteresowanych

Na SKOS został udostępniony program w języku Racket (w wariancie z typami), który przetwarza programy napisane w prostym języku z let-wyrażeniami, wyrażeniami warunkowymi i funkcjami rekurencyjnymi (pierwszego rzędu) do programów dla programowalnego kalkulatora RPN. Program ten został wykorzystany do opracowania przykładów podanych powyżej. Zachęcam zainteresowane osoby do napisania własnych programów przykładowych.

Skrypt testujący

Do testowania rozwiązania można użyć poniższego skryptu Lua. Skrypt ten zawiera, oprócz kodu sterującego układem, przykładowe programy wymienione powyżej oraz funkcję kodującą program zgodnie z opisanym kodowaniem. Wyboru przykładu do uruchomienia można dokonać przez zmianę pierwszej linijki po komentarzu `skrypt testujący`.

```lua
-- programy przykładowe
example_g1 = {5, 'g'}
example_g2 = {0, 'g'}
example_g3 = {5, '-', 'g'}
example_sp = {1, 2, 's', 'p'}
example_l1 = {3, 4, 0, 'l', 's', 'p', 's', 'p'}
example_l2 = {3, 4, 1, 'l', 's', 'p', 's', 'p'}
function example_factorial(x)
  return {4, x, 6, 'j', 32, 'j', 0, 'l', 'g', 3, '*',
    14, '+', 'j', 1, 28, 'j', 0, 'l', 27, 2, 'l', 1,
    '-', '+', 6, 'j', '*', 's', 'p', 's', 'j'}
end
function example_power(x, y)
  return {5, x, y, 7, 'j', 37, 'j', 0, 'l', 'g', 3,
    '*', 15, '+', 'j', 1, 31, 'j', 1, 'l', 30, 3,
    'l', 3, 'l', 1, '-', '+', 7, 'j', '*', 's', 'p',
    's', 'p', 's', 'j'}
end
function example_gcd(x, y)
  return {5, x, y, 7, 'j', 68, 'j', 1, 'l', 1, 'l',
    '-', '+', 0, 'l', '-', 'g', 's', 'g', '+', 'g', 4,
```

```
    '*', 26, '+', 'j', 1, 'l', 62, 'j', 1, 'l', 1, 'l',
    '-', '+', 'g', 9, '*', 42, '+', 'j', 49, 1, 'l', 3,
    'l', 7, 'j', 62, 'j', 62, 2, 'l', 2, 'l', '-', '+',
    2, 'l', 7, 'j', 's', 'p', 's', 'p', 's', 'j'}
end
-- kodowanie programów
instruction_encodings = {
  ['g'] = 0, ['-'] = 1, ['+'] = 2, ['*'] = 3,
  ['s'] = 4, ['l'] = 5, ['p'] = 6, ['j'] = 7
}
function encode_program(arr)
  local out = {}
  local i = 1
  while arr[i] do
    if type(arr[i]) == "number" then
      out[i] = arr[i]
    elseif instruction_encodings[arr[i]] then
      out[i] = 0x8000 + instruction_encodings[arr[i]]
    else
      error("Invalid symbol in program: " .. arr[i])
    end
    i = i + 1
  end
  out[i] = 0xffff
  return out
end
-- skrypt testujący
arr = encode_program(example_factorial(3))
sim.setinput("wr", 0)
sim.setinput("start", 0)
sim.setinput("nrst", 0)
sim.sleep(100)
sim.setinput("nrst",1)
sim.sleep(100)
assert(sim.getoutput("ready"):ishigh(), "Reset failed!")
sim.wait(sim.posedge("clk"))
sim.setinput("wr", 1)
for i, v in ipairs(arr) do
  sim.setinput("addr", i-1)
  sim.setinput("datain", v)
  sim.wait(sim.posedge("clk"))
end
sim.setinput("wr", 0)
sim.setinput("start", 1)
sim.wait(sim.posedge("clk"))
sim.setinput("start", 0)
sim.sleep(10)
assert(sim.getoutput("ready"):islow(), "Start failed!")
while sim.getoutput("ready"):islow() do
  sim.wait(sim.posedge("clk"))
  sim.sleep(10)
end
print("Program result: " .. sim.getoutput("out"):tointeger())
```