```
function CompileDepth(c, S)
begin
  switch S

  case [nothing]:
    return ({}, {})

  case [x = τ]: # actions
    return ({}, {})

  case [ℓ: pause(C)]: # pause
    return ({}, {ℓ ∧ C ∧ susp_S(C) ⇒ next(ℓ) = true})

  case [if (γ) { S1 } else { S2 }]: # conditional
    (A1^data, A1^ctrl) := CompileDepth(c, S1)
    (A2^data, A2^ctrl) := CompileDepth(c, S2)
    return (A1^data ∪ A2^data, A1^ctrl ∪ A2^ctrl)

  case [S1; S2]: # sequence
    (A1^data, A1^ctrl) := CompileDepth(c, S1)
    (A2^data, A2^ctrl) := CompileSurface(c, term_S1, S2)
    (A2^data, A2^ctrl) := CompileDepth(c, S2)
    return (A1^data ∪ A2^data, A1^ctrl ∪ A2^ctrl)

  case [S1 || S2]: # parallel threads
    (A1^data, A1^ctrl) := CompileDepth(c, S1)
    (A2^data, A2^ctrl) := CompileDepth(c, S2)
    return (A1^data ∪ A2^data, A1^ctrl ∪ A2^ctrl)

  case [suspend { S' } when(γ)]:
    return CompileDepth(c, S')

  case [ℓ: immediate suspend { S' } when(γ)]:
    (A^data, A^ctrl) := CompileDepth(c, S')
    return (A^data, A^ctrl ∪ {strt ∧ γ ⇒ next(ℓ) = true})

  case [abort { S' } when(γ)]:
    return CompileDepth(c, S')

  case [immediate abort { S' } when(γ)]:
    return CompileDepth(c, S')

  case [clock (C) { S' }]: # clock declaration
    return CompileDepth(C, S')
end
```